NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

RELIABLE CONTENT DELIVERY USING PERSISTENT DATA SESSIONS IN A HIGHLY MOBILE ENVIRONMENT

by

Periklis K. Pantoleon

March 2004

Thesis Advisor:	Wen Su
Thesis Co-Advisor:	John Gibson

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** March 2004	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE: Reliable Content Delivery Using Persistent Data Sessions in a Highly Mobile Environment			**5. FUNDING NUMBERS**
6. AUTHOR(S) Periklis K. Pantoleon			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release, distribution is unlimited			**12b. DISTRIBUTION CODE**

13. ABSTRACT ***(maximum 200 words)***

Special Forces are crucial in specific military operations. They usually operate in hostile territory where communications are difficult to establish and preserve, since the operations are often carried out in a remote environment and the communications need to be highly mobile. The delivery of information about the geographical parameters of the area can be crucial for the completion of their mission. But in that highly mobile environment, the connectivity of the established wireless networks (LANs) can be unstable and intermittently unavailable.

Existing content transfer protocols are not adaptive to volatile network connectivity. If a physical connection is lost, any information or part of a file already retrieved is discarded and the same information must be retransmitted again after the reestablishment of the lost session. The intention of this Thesis is to develop a protocol in the application layer that preserves the already transmitted part of the file, and when the session is reestablished, the information server can continue sending the rest of the file to the requesting host. Further, if the same content is available from another server through a better route, the new server should be able to continue to serve the content, starting from where the session with the previous server ended.

14. SUBJECT TERMS File Transfer Protocol, Partial File Retrieval, SFTP, TFTP, HTTP			**15. NUMBER OF PAGES** 207
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release, distribution is unlimited

RELIABLE CONTENT DELIVERY USING PERSISTENT DATA SESSIONS IN A HIGHLY MOBILE ENVIRONMENT

Periklis K. Pantoleon
Lieutenant, Hellenic Navy
B.S., Hellenic Naval Academy, 1995

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
March 2004

Author: Periklis K. Pantoleon

Approved by: Wen Su
Thesis Advisor

John Gibson
Thesis Co-Advisor

Peter J. Denning
Chairman, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Special Forces are crucial in specific military operations. They usually operate in hostile territory where communications are difficult to establish and preserve, since the operations are often carried out in a remote environment and the communications need to be highly mobile. The delivery of information about the geographical parameters of the area can be crucial for the completion of their mission. But in that highly mobile environment, the connectivity of the established wireless networks (LANs) can be unstable and intermittently unavailable.

Existing content transfer protocols are not adaptive to volatile network connectivity. If a physical connection is lost, any information or part of a file already retrieved is discarded and the same information must be retransmitted again after the reestablishment of the lost session. The intention of this Thesis is to develop a protocol in the application layer that preserves the already transmitted part of the file, and when the session is reestablished, the information server can continue sending the rest of the file to the requesting host. Further, if the same content is available from another server through a better route, the new server should be able to continue to serve the content, starting from where the session with the previous server ended.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I. INTRODUCTION 1
- **A. PROBLEM STATEMENT 1**
- **B. RESEARCH QUESTIONS 1**
- **C. SCOPE AND METHODOLOGY 1**
  - **1. Scope 1**
  - **2. Methodology 2**
  - **3. Assumptions and Limitations 2**
    - ***a. Assumptions 2***
    - ***b. Limitations 2***
- **D. ORGANIZATION 3**

II. BACKGROUND 5
- **A. EXISTING FILE TRANSFER PROTOCOLS 5**
  - **1. File Transfer Protocol (FTP) 5**
  - **2. Trivial File Transfer Protocol (TFTP) 8**
  - **3. Simple File Transfer Protocol (SFTP) 11**
  - **4. Hypertext Transfer Protocol (HTTP) 13**
- **B. FILE TRANSFER IN MOBILE/ HANDHELD DEVICES 15**

III. APPLICATION DESIGN 17
- **A. REQUIREMENTS 17**
  - **1. Functional Requirements 17**
    - ***a. User Interface 17***
    - ***b. File Transfer 19***
    - ***c. File Database 20***
    - ***d. File Authentication 20***
    - ***e. PFTP Server Availability 20***
    - ***f. Client-Server Communication Protocol and User Interaction 20***
  - **2. Non Functional Requirements 23**
    - ***a. User Requirements 23***
    - ***b. Hardware 23***
    - ***c. Software 23***
- **B. UML DIAGRAMS 24**
  - **1. Use Case Diagrams 24**
  - **2. Class Diagrams 25**
    - ***a. User Interface Classes 25***
    - ***b. Desktop/Laptop PFTP Application Version 29***
    - ***c. Pocket PC PFTP Application Version 30***
  - **3. Activity Diagram 31**
  - **4. Flow Sequence Diagrams 32**
    - ***a. Add a File in Database 32***

b. Remove a File from Database 33
c. Transfer a File 34
d. Cancel File Transfer and Save Partial Data Use Case 35
e. Continue a Previous Partial File Transfer 36

IV. APPLICATION DEVELOPMENT 37
A. DEVELOPMENT TOOLS 37
B. USER INTERFACE 37
1. PFTP Server 37
2. PFTP Client 40
C. FUNCTIONALITY IMPLEMENTATION 41
1. File Transfer Failure Recovery 41
2. File Authentication 41

V. TESTING 43
A. TESTING NETWORK DESCRIPTION 43
1. Considerations-Limitations 43
2. Testing Network 43
B. TESTING SCENARIOS 44
C. TESTING RESULTS 46

VI. CONCLUSIONS AND FUTURE WORK 55
A. SUMMARY 55
B. FURTHER WORK 56
1. Communication Protocol Design 56
2. Application Development 56

APPENDIX. CLASS SOURCE CODE 57

LIST OF REFERENCES 189

INITIAL DISTRIBUTION LIST 191

LIST OF FIGURES

Figure 1. The use model of the FTP protocol....6
Figure 2. The alternative use model of the FTP protocol....6
Figure 3. The data block header in the block transmission mode of FTP....7
Figure 4. Compression of data bytes in the compressed mode of FTP....8
Figure 5. Data transfer sequence in TFTP....9
Figure 6. Messages structure of TFTP....10
Figure 7. An example of a file transfer connection in SFTP....13
Figure 8. The types of request fulfillment in HTTP protocol....14
Figure 9. Preview of the client's user interface in PFTP application....18
Figure 10. Preview of the Pocket PC version client's user interface....18
Figure 11. Preview of the server's user interface in PFTP application....19
Figure 12. The communication scheme of PFTP application....20
Figure 13. The communication scheme of PFTP application....21
Figure 14. The communication protocol of PFTP application....22
Figure 15. The use case diagram of the PFTP application....24
Figure 16. The class diagram of the PFTP Desktop/Laptop Client user interface....26
Figure 17. The class diagram of the PFTP Pocket PC Client user interface....27
Figure 18. The class diagram of the PFTP Server user interface....28
Figure 19. The class diagram of the PFTP application....29
Figure 20. The class diagram of the Pocket PC PFTP Application version....30
Figure 21. The activity diagram of the PFTP application....31
Figure 22. The sequence diagram of "Add a file in database" use-case....32
Figure 23. The sequence diagram of "Remove a file in database" use case....33
Figure 24. The sequence diagram of "Transfer a file" use-case....34
Figure 25. The sequence diagram of "Cancel file transfer and save partial data" use case....35
Figure 26. The sequence diagram of "Continue a previous partial file transfer" use case....36
Figure 27. User interface of the PFTP Server application....37
Figure 28. User interface when the **Server** menu item is selected....38
Figure 29. User interface when the server starts....38
Figure 30. User interface when the **Database** menu item is selected....39
Figure 31. User interface when the **Database** menu item is selected....39
Figure 32. The user interface for the client side in the desktop/laptop version....40
Figure 33. The user interface for the client side in the desktop/laptop version....40
Figure 34. Code segment to produce the file hash value....42
Figure 35. The testing network of PFTP application....44
Figure 36. The response of server side in the SUIS-1 scenario....46
Figure 37. The response of server side in the SUIS-2 scenario....47
Figure 38. The message when a already existing file is added in file database....47
Figure 39. The message when no previous partial file transfer exists....48

Figure 40. The same message as Figure 5 in Pocket PC device 48
Figure 41. The message when not all the file transfer option are selected.....49
Figure 42. The same message as Figure 7 in Pocket PC environment.....49
Figure 43. Paused state of the user interface after a connection loss (laptop) 50
Figure 44. Paused state of the user interface after a connection loss (Pocket PC).....50
Figure 45. The system output that shows the reconnection procedure 51
Figure 46. The user interface when the file is transfer is completed in the laptop version.....51
Figure 47. The user interface when the file transfer is completed in the Pocket PC version.....52
Figure 48. Auto server search after a connection loss in auto server mode 53

LIST OF TABLES

Table 1. Commands and replies of a SFTP communication protocol 12
Table 2. The testing scenarios for the PFTP application 45

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

This thesis is dedicated to my family who, even though they were far away, they had their own way to support and encourage me through all this demanding process.

I would like to express my appreciation to my advisors, Professor Su Wen and John Gibson. Without their support and their supervision, this thesis would not have been possible.

I also thank my country and the Greek Navy who gave the opportunity to have this postgraduate degree in Computer Science at Naval Postgraduate School.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. PROBLEM STATEMENT

One of the major problems during the operation of a wireless mobile LAN, is the reliability of the content delivery under difficult networking conditions. Currently, an error or connection loss during a data transfer session can only be handled by retransmitting the data. However, in the wireless network established to support Special Forces teams in a highly mobile environment, the timely delivery of information is essential for the accomplishment of the mission. In this case, a more rapid and dynamic data recovery must be achieved, so that the interrupted data transfer session can be continued from the point it stopped.

B. RESEARCH QUESTIONS

Addressing the above problem, the current research will address the following questions:

- What is an appropriate design for a file transfer application so that partial file retrieval can be achieved?
- What is an example of a high-layer communication protocol that can be used in this application?
- How can this application design support the use of mobile devices?
- How can this communication model be flexible when the availability of the information servers varies?

C. SCOPE AND METHODOLOGY

1. Scope

The scope of the thesis is to develop a client–server file transfer application named PFTP (Partial File Transfer Protocol) to demonstrate a possible solution to the problem of a lack of persistent data sessions in wireless mobile networks and LANs. For this purpose, a prototype communication protocol between the client and the server will be designed to achieve dynamic partial file retrieval in the event of connection loss. The goal is to produce an application user interface that visualizes the partial file retrieval process in real time.

2. Methodology

This thesis research will follow the methodology below:

- Search for partial file transfer capabilities in the existing file transfer protocols
- Specify the application's requirements for both the user interfaces and the communication protocol
- Design and develop a file transfer application using Java programming.
- Test the application in a wireless environment, against several scenarios simulating real situations that can occur.

3. Assumptions and Limitations

a. Assumptions

- Programming Language. The development of the application will be done with Java Technology programming. As Java programming support becomes more and more powerful, it contains the tools necessary to support the development of both user interface and communication protocol specific components for the purpose of the thesis research.
- Mobile Device. The mobile platform for the mobile client of this application is a Pocket PC technology device, which possesses a more user friendly file management system.

b. Limitations

- Security. Except for the file authentication that is achieved with the file hash value exchange procedure during the transfer process, the security aspect of this protocol is not covered in the manner in which it can support the military use of the application. A secure tunneling design would be appropriate to encapsulate each session between the client and server.
- File Database Management. The application server has limited functionality in the file database management. The server side creates a local file database that holds the files that are decided by the server administrator to be available for the clients before it runs.
- Scalability. Even though the multithreading approach is used to design the server side of the application, this thesis research does not address the capability to support a large number of users that produce heavy traffic on a single server.

D. ORGANIZATION

The covered material is organized into the following chapters in order to fulfill the objectives of this thesis. Chapter II will refer to all the available file transfer protocols and how a built in capability exists in some of them for partial file retrieval. Chapter III will cover the application design consisting of the requirements and the UML diagrams. Chapter IV will describe the development process, referring to the developments tools used, the user interface functionality and how the most important features of the application are implemented for both the client and server side. Chapter V will address the testing phase of the application development. Specific scenarios will test the manner in which applications respond to any file transfer case. Finally, Chapter VI will present recommendations, conclusions, and further work to be done in the establishment of persistent data sessions in file transfers within a mobile LAN.

THIS PAGE INTENTIONALLY LEFT BLANK

II. BACKGROUND

This chapter refers to some of the most known file transfer protocols and how some have features that can enable a form of partial file recovery. Also, it contains information about the mobile devices' networking characteristics, as far as the file transfer protocols they are using.

A. EXISTING FILE TRANSFER PROTOCOLS

1. File Transfer Protocol (FTP)

FTP is the oldest of the current file transfer protocols, introduced for the first time in 1971 [1] .The last specification for this protocol is described in Ref.[2]. FTP is a user–level protocol for file transfer among hosts and terminal Interface Message Processors (IMP) and it uses the Data Transfer Protocol (DTP)[3] to transfer the file data.

The main goals of the FTP protocol are to:

- Use the remote file storage between hosts conveniently
- Promote the sharing of files
- Encourage its use by computer programs. Even though FTP can be used by a PC user, it is designed for use by programs
- Keep the user safe from any difference in file storage systems among host operating systems
- Achieve reliable and efficient data transfers

The communication model of FTP is based on the establishment of two bidirectional connections between the client and the server, the ***Control Connection*** using port 21 for the exchange of the FTP commands and replies using Telnet protocol, and the ***Data Connection*** using port 20 for the data transfers using the TCP protocol (Figure 1). The protocol interpreter of the client side is that which initiates the control connection using the Telnet protocol. After the initiation of the control connection, the user protocol interpreter generates and sends the FTP commands to the server that uses standard replies to respond through the control connection.

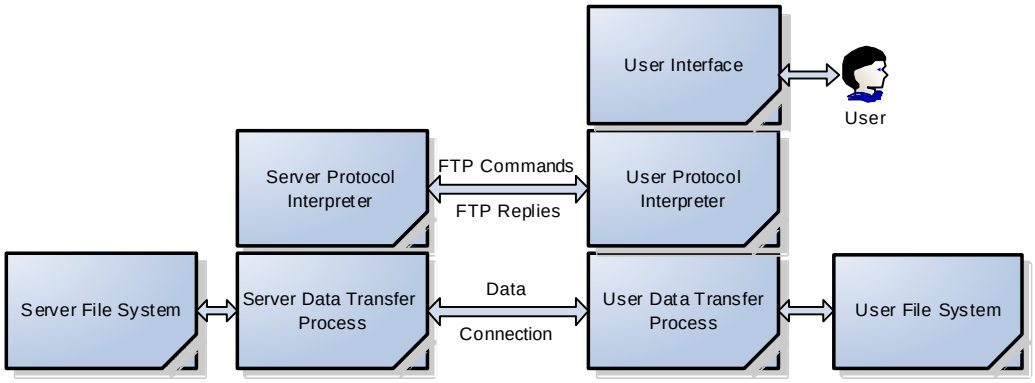

Figure 1. The use model of the FTP protocol

FTP commands specify the parameters of the data connection such as the transfer mode, data port, file structure and representation type, and what kind of file operations are requested such as retrieve , store and delete. The server then initiates the data connection to a data transfer port that is specified by the user, and sends the data applying the parameters. The client must listen to that port that also can be different from the port client used to initiate the control connection. Except for this client–server data transfer, FTP provides the capability so that the user on the client side can tell the server to send the data to another FTP server host (Figure 2).

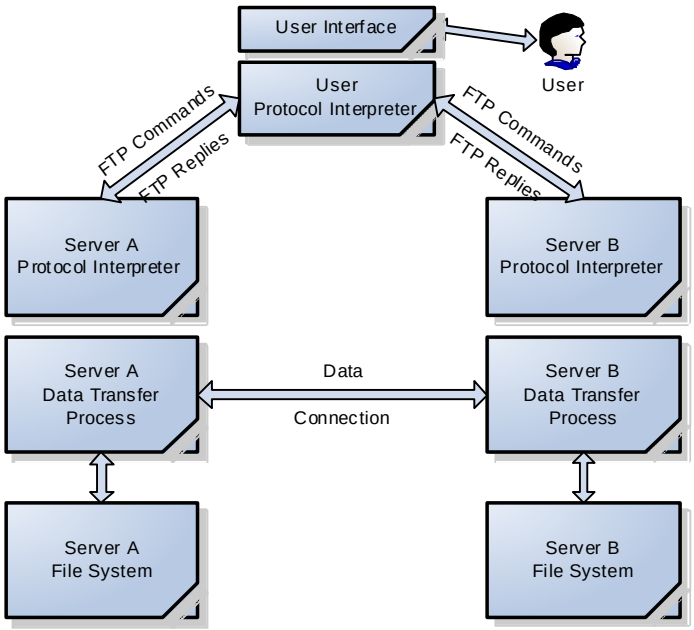

Figure 2. The alternative use model of the FTP protocol

In that case, control connections are initiated by the client protocol interpreter with both protocol interpreters of the two server hosts but the data connection is established between the two servers for exchanging the file data. The server that sends the data in both cases mentioned above is responsible for initiating, maintaining, and closing the data connection.

Three transmission modes must be supported by every implementation of the FTP protocol for the file data transfer to occur. They are the stream mode, the block mode and the compressed mode. In all modes, an end-of-file (EOF) marker, depending on the file structure, must explicitly or implicitly indicate the end of the data transfer or the close of the data connection is identified as the end of the file indicator.

In the ***stream mode***, the file data are transmitted as a stream of bytes without excluding the record structured files. In the ***block mode***, files are a series of data blocks that have a tree byte header that contain a one–byte descriptor code and a two–byte count field (Figure 3).

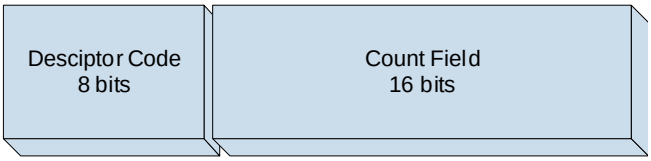

Figure 3. The data block header in the block transmission mode of FTP

The descriptor code field indicates the end of file (EOF), the end of record (EOR), the restart marker or the suspected for errors data marker with bit flags. The count field indicates the total length in bytes of the data blocks. In ***compressed mode***, the information that must be sent is the actual data in a form of byte string, the compressed data that consist of replications or filler and control information [2]. The n data bytes are compressed to one replicated byte (Figure 4).

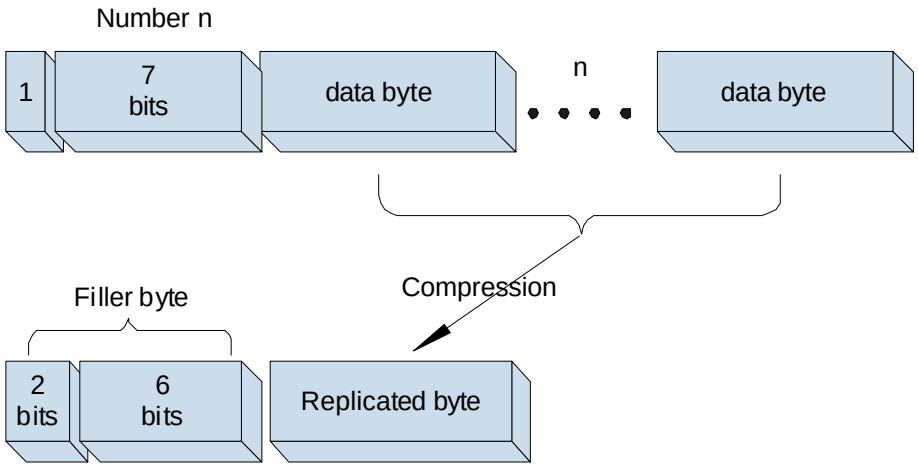

Figure 4. Compression of data bytes in the compressed mode of FTP

In general, FTP does not have mechanisms to detect if any bit loss or the data received is scrambled. This kind of low level error control is handled by the underlying TCP. However, in the case of a long data transfer, such as in big files, and in the ***block*** or ***compressed*** transfer mode, a ***restart procedure*** exists that is trying to protect the user from the failures of hosts or the underlying network during the data transfer. The procedure is based on special ***market codes*** that represent a data–unit counter (bit, byte or record), that the sender inserts in the data stream at some points. Those markers, also known as ***checkpoints***, in the case of a system failure, act as milestones in the transfer process and can be used by the user to ask for data recovery by sending a restart command, with the marker code identified.

However, even though this error recovery feature exists, Ref. [2] does not mention any implementation details and an agreement between FTP client/server vendors about a common format for the restart markers does not exist, and therefore, they can be used effectively.

2. Trivial File Transfer Protocol (TFTP)

TFTP is a very simple protocol that is used to transfer files and is described in Ref [4] and revised by Ref [5]. It is designed to be on top of the UDP protocol even though other transport protocols are not excluded. To keep its implementation as simple as possible, TFTP does not have features that can be found in FTP protocol such as

directory listing and user authentication. Files can be transferred with three modes, netascii (8 bit ascii mode) which is a standard for transferring text files, octec mode which is for transferring binary files, and mail mode which is for mail[1].

The communication protocol starts with the TFTP client sending a ***read*** or ***write request*** on port 69 of the TFTP server to read or write a file. If the server accepts the request, then a connection is established and the data are sent in fixed blocks of 512 bytes. Each non-terminal data packet must be acknowledged before the sender sends the next one. If a timeout period is passed without acknowledge reception for a packet, this packet is resent. Packet reception with a length less than 512 bytes indicates a terminal file data block and the termination of a file transfer (Figure 5).

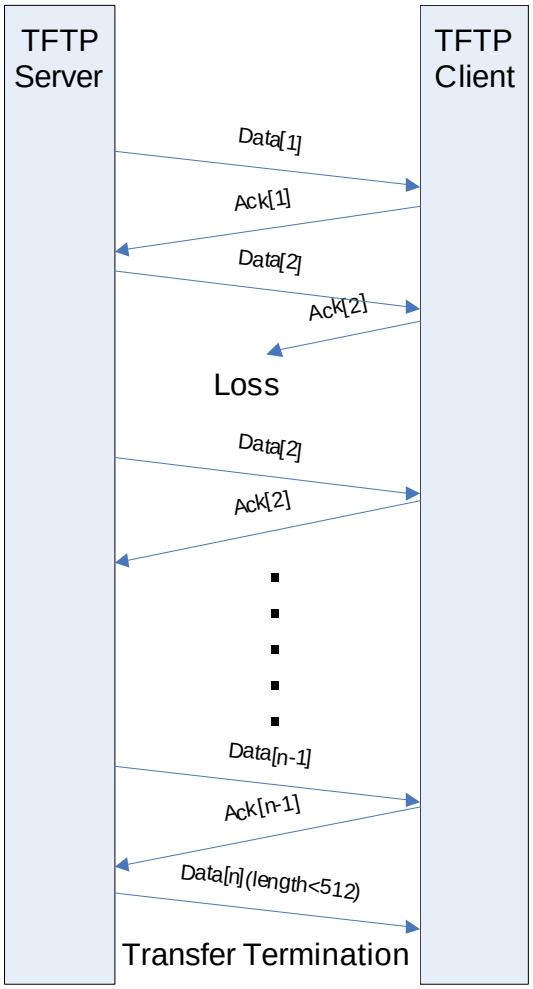

Figure 5. Data transfer sequence in TFTP

[1] Mail mode is not implemented in commercial TFTP applications.

In case of a lost packet, the receiver sends its last transmitted packet to indicate this loss and to cause the sender[2] to resend it.

There are four types of messages that are exchanged between the client and the server:

- Read request message
- Write request message
- Data message
- Acknowledge message
- Error message

All message packets have a 2 byte front opcode that identifies the format of TFTP messages and then the data depending on the message type. Figure 6 shows the packet structure of different message packets.

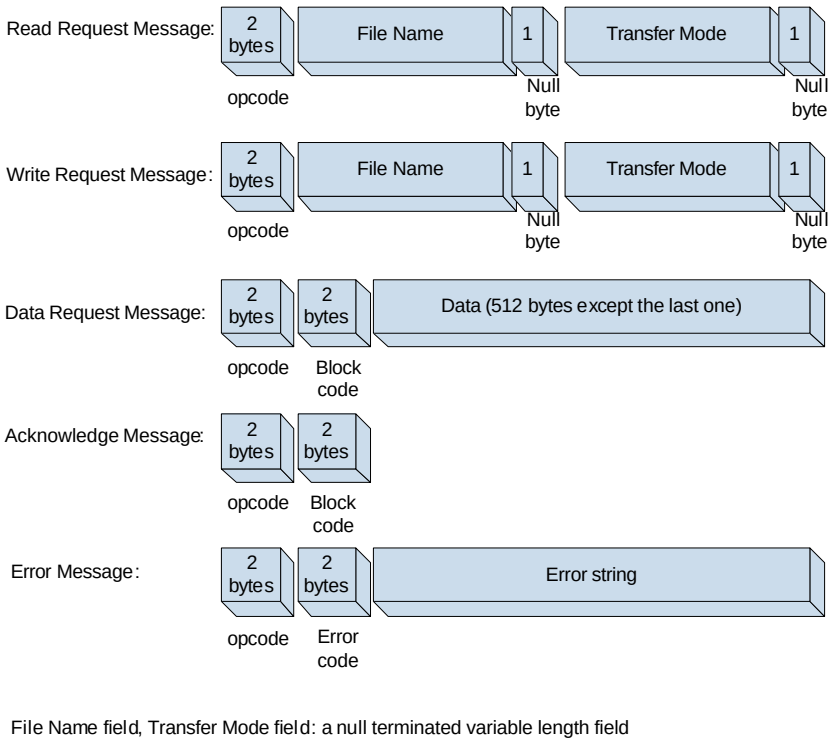

Figure 6. Messages structure of TFTP

[2] The sender in TFTP must keep one previous sent data packet in case of packet loss.

When an error occurs (an error message received) during the file transfer, the designed reaction of the TFTP protocol is to terminate the connection. The error message is not acknowledged, so when an error message has been lost, the receiver uses a timeout period to detect this connection termination. In this protocol after an abnormal termination, no procedure exists to retrieve the lost part of a file after a connection failure during transfer.

3. Simple File Transfer Protocol (SFTP)

The simple file transfer protocol (SFTP) is a protocol designed to be more useful than TFTP but not as powerful as FTP [6]. SFTP uses one TCP connection to transfer the files compared to one UDP connection of the TFTP protocol and the two TCP connections of FTP. This characteristic gives the SFTP the ability to support:

- File transfers
- Directory listing and changing
- User access control
- File deleting and renaming

The SFTP communication protocol starts with the client side connecting to the server on port 115 via TCP. Then, the client sends commands to the server and waits for responses. The SFTP commands and replies are in the form of:

Commands : **<4 ASCII character command> [<space> <arguments>] <null>**

Replies : **<1 ASCII response character> [<ASCII message string>] <null>.**

In Table 1, all the possible commands and replies are listed with a short description. Also, Figure 7 shows an example of a command–reply exchange that leads to file transfer.

COMMAND	**DESCRIPTION**
USER	The user ID on the server system
ACCT	The account to use on the server system.
PASS	The password on the server system
TYPE	The mapping of the stored file to the transmission byte stream is controlled by the type. The default is binary if the type is not specified.
LIST	Lists the directory contents.
CDIR	Changes the current directory on the server host to the given one.
KILL	Deletes the file from the server system.
NAME	Renames the file on the server system.
DONE	Tells the server system that the process is finished.
RETR	Requests that the server system send a specified file.
STOR	Tells the server system to receive a file and save it under the name given.
REPLY	**DESCRIPTION**
+	Success.
-	An error has occurred
!	Logged In
<space>	Number

Table 1. Commands and replies of a SFTP communication protocol

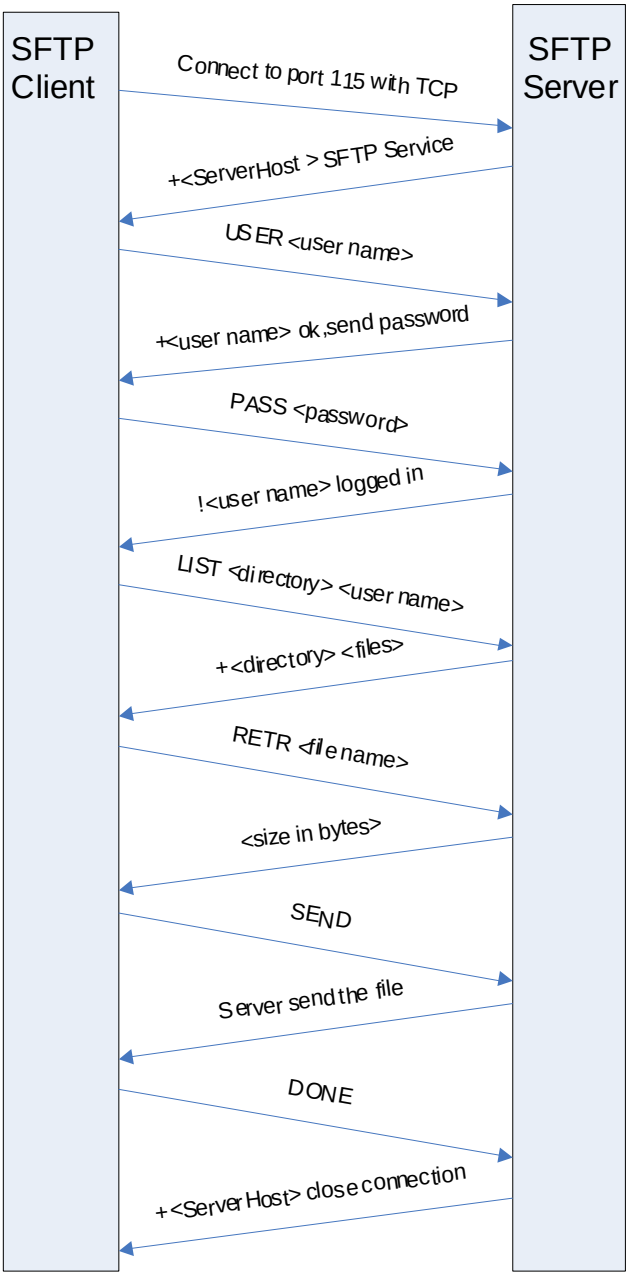

Figure 7. An example of a file transfer connection in SFTP

In SFTP, no way exists to retrieve the remaining part of the file if the traditional TCP connection fails during the transmission of file bytes.

4. Hypertext Transfer Protocol (HTTP)

The Hypertext Transfer Protocol (HTTP) as described in Ref. [7] is an application–level protocol that is used for the transfer of hypermedia information on the

Internet. HTTP is a request /response type of transfer protocol. The client opens a connection with the server and sends a request. The server responds to this request and closes the connection. These connections can be persistent by choosing TCP as the underlying protocol.

The client side user initiates the communication protocol connecting to port 80 of the HTTP server host and sends a request. The client request can be fulfilled by three general cases. In the first case, the client request reaches the origin server through a single connection and the response comes from the same connection and server (Figure 8(a)). In the second case, there are some intermediaries, such as gateways and proxies, between the client and the origin server that maybe do not understand the message but act as a relay point (Figure 8(b)).

The third is the case when the HTTP request and responses between the client user and origin server can be cached and reused to lower the traffic to the origin server (Figure 8(c)).

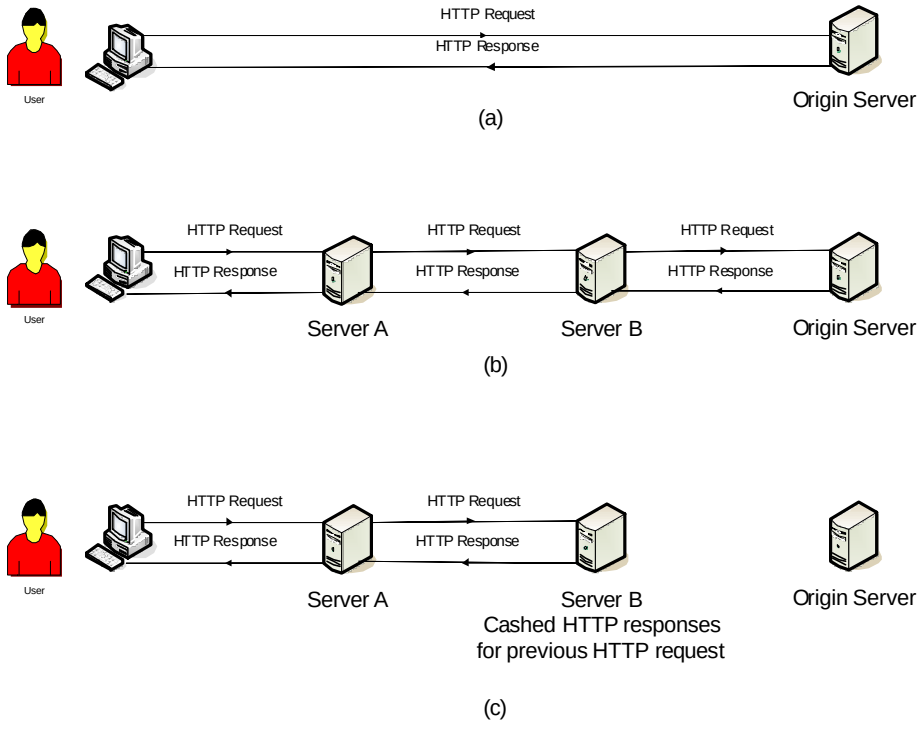

Figure 8. The types of request fulfillment in HTTP protocol

HTTP protocol is designed so that it is possible to request one or more parts (or ***ranges***) of a hypermedia information file, instead of the entire file. For that purpose, special ***range request headers*** can be used and the HTTP servers, as is proposed by Ref.[5], ought to support range request headers when possible, because it is an efficient way to recover from the partially failed transfer of large files.

B. FILE TRANSFER IN MOBILE/ HANDHELD DEVICES

By the time people become accustomed to having a mobile or a handheld device which helps them stay connected with other people or organize their business or family life, new capabilities have been examined to bring those devices to the Internet and inter–networking community. Newer enhancements in mobile phones bring the Internet–ready cellular phones or Web phones in use, making them able to access the Web, running micro–browsers. Wireless enabled handheld devices, such as Palm or Pocket PCs, having more computational power and memory, can also store data locally in the form of records or regular files respectively. Due to the need for Internet access, the most popular file transfer protocols among these devices is HTTP, but software vendors have not enabled any partial recovery capabilities, where they are available and the only way to handle a failed file transfer connection is to resend the file.

THIS PAGE INTENTIONALLY LEFT BLANK

III. APPLICATION DESIGN

This chapter presents the initial requirements that the application addresses. In Section B, the UML diagrams show how the application is designed in greater detail.

A. REQUIREMENTS

1. Functional Requirements

a. User Interface

The PFTP application will provide a user interface for both the client-side and server–side modules. In this initial version, and for the purpose of the thesis research, the user interface must provide four main services to the client-side user. They are:

- Control of the file transfer communication. The user must be able to control different options of the file transfer, such as the file name, server selection/configuration mode, and the packet size. Also, there must be an option for the user to stop or cancel the file transfer at any time.
- View of the file transfer progress. A file transfer progress bar with the percentage of the transfer finished will be sufficient.
- View of the file transfer status. Useful information must be accessible by the user during the file transfer.
- Visual preview of the file during transfer in the case of an image file.

These services can be represented as separate panels on the main user interface frame. Figure 9 shows a primitive preview of an example of a PFTP client side user interface.

Figure 9. Preview of the client's user interface in PFTP application

In the Pocket PC version of the PFTP client, it is not possible to show all the above-mentioned panels concurrently, due to the screen size of this device. In that case, the control panel will be the main panel and the image progress panel will appear when the file is downloading.

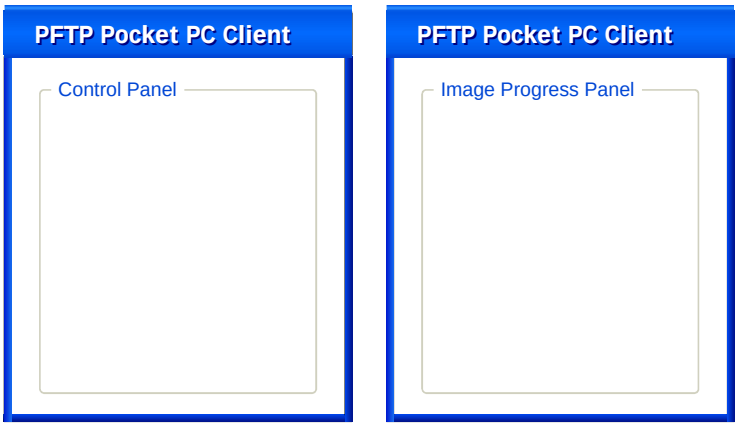

Figure 10. Preview of the Pocket PC version client's user interface

The PFTP server-side user interface must support limited control and administration functionality for the server application. These functions will be:

- Limited server application control. A simple starting and stopping functionality will be available.
- File database management. The server has a local file database that keeps the files available to clients for transfer. The server application user can add or remove files from the database.
- Monitoring the server's operation. When the server starts, a panel must be present to show the operation status and some connection failure statistics.

The first two of the above functionalities can be available as a menu item in a menu bar at the top of the user interface. Under this a panel can be placed displaying a logo before the server application starts or the server-monitoring panel after server initialization. Figure 11 shows the intentional user interface preview.

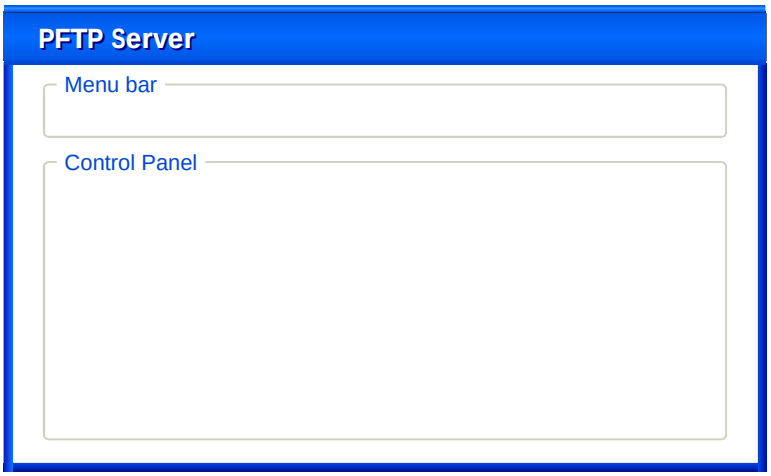

Figure 11. Preview of the server's user interface in PFTP application

b. File Transfer

The file transfer from the PFTP Server to the PFTP Client results from a client-side user request. The user selects the preferred features of the file transfer (file name, PFTP server, packet size), and asks to download it. The server sends the file as packets (arrays of bytes) that the client collects until it receives all of them. The client application keeps track of the number of packets received, and in case of a failure in the connection, keeps attempting to reconnect with the server and send a request to retrieve the rest of the file packets.

c. File Database

The server of this application must keep a file database to store the files available to the clients for transfer. The server cannot start unless the user first adds files to the database. Any file can be removed at any time from the database but the application stops running when the database becomes empty. Client-side user is advised when the server file database is empty.

d. File Authentication

After a connection failure and during the reconnection period, the client will request the same file either from the same or another PFTP server. In order for the server to know if it has the right file (same name, modification, or content) with this name, it must use a value that uniquely identifies it. A hashing algorithm such as MD5 can produce a hash value for the file that is sufficient for this authentication check. That value will be sent by the server to the client after the initial file request, so that it can be available later, to be included in the partial file recovery request to another PFTP server, in the event of a connection loss.

e. PFTP Server Availability

PFTP client-side keep a list of the PFTP server IP addresses to try during the reconnection. For better performance a getServer() function should be called that can search for a valid server to connect to, which is beyond the scope of the thesis research Also, the client-side user can add any additional PFTP server available at a later time.

f. Client-Server Communication Protocol and User Interaction

PFTP is an application layer protocol and it is based on a client-server communication scheme. For the file transfer process, an application layer communication protocol must be established. The TCP protocol is the underling protocol for every connection and data transfer between the client and server (Figure 12).

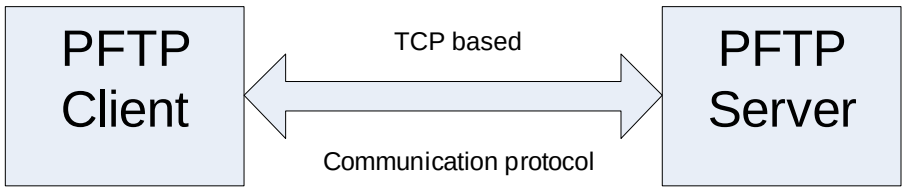

Figure 12. The communication scheme of PFTP application

This communication protocol is shown in Figure 13 and starts with the PFTP server running and waiting for clients at port 6789. When the user opens the PFTP client application, it must choose a PFTP server manually or from a list of already existing servers in order to retrieve the list of files available for transfer. In the predefined server case, an Auto server mode option exists which can be selected when the user wants the application to choose a predefined PFTP server randomly during either the initial available file retrieval or the partial file retrieval process after a connection loss. When the file list is retrieved, a user selects a file name and the packet size and forms a file request packet that is sent to the server. The request packet fields are shown in Figure 14.

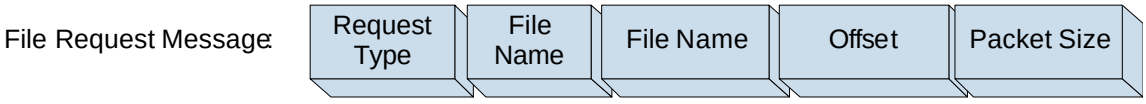

Figure 13. The communication scheme of PFTP application

The server checks if it has the file and, if it does, sends the file size and the hash value of the file back to client. Then the server starts reading the file data and sends it in packets to the client. If, during the packet transmission, a connection failure occurs, the client side generates partial-file-request packet, setting the offset field value with the next expected packet counter and the hash field value with the file hash received at the start of the file transfer. Then, the client attempts to reconnect with the same server or, if the Auto server mode selected, with the next available PFTP server, and when it is connected, sends the prepared partial-file-request packet.

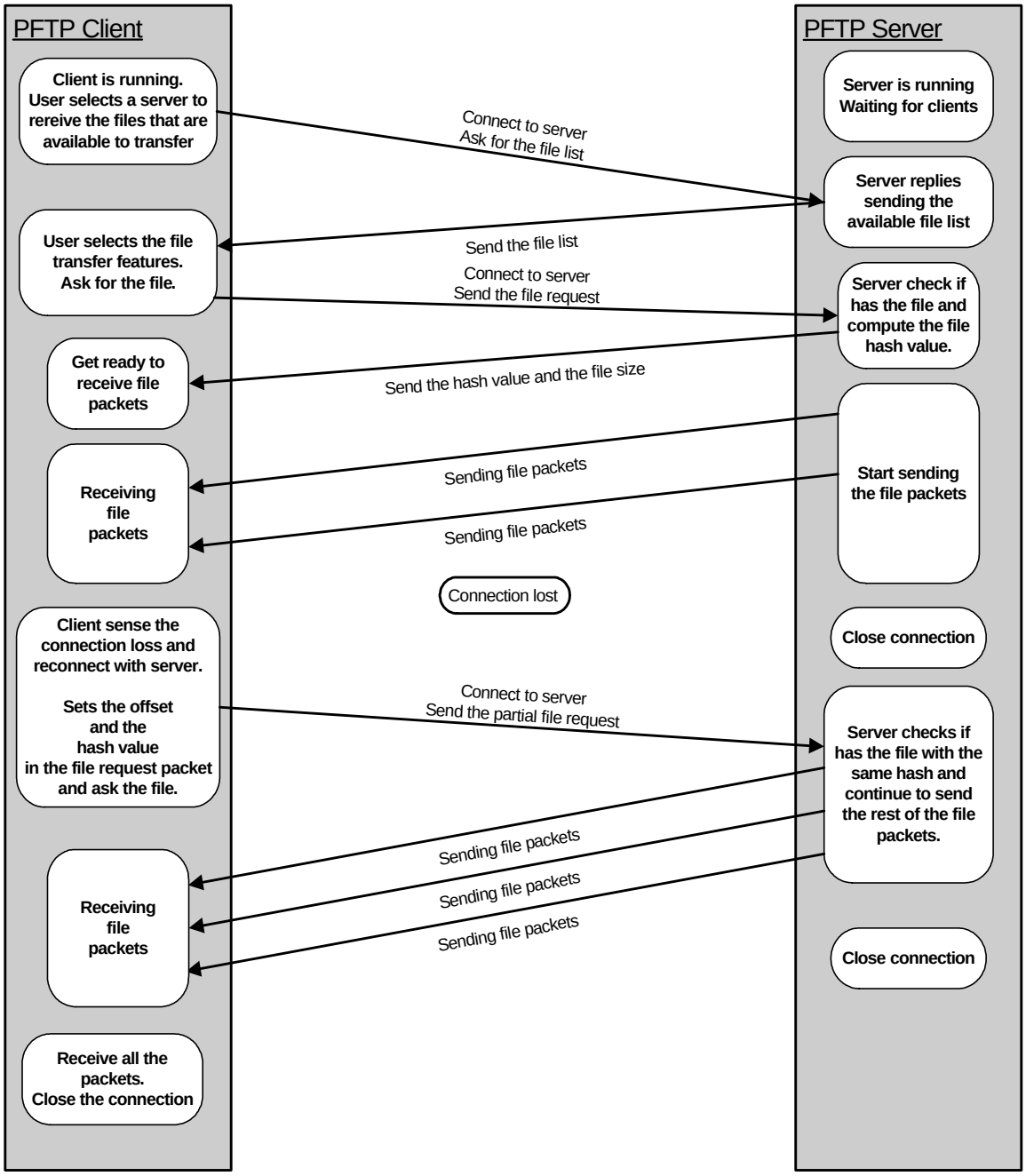

Figure 14. The communication protocol of PFTP application

The server that receives the packet, if the offset value is greater that zero, checks if it has the same file with the same hash value and continues sending the remaining data in packets to the client. Each file connection loss causes the same partial-transfer retrieval scheme until the client receives all the file packets and both the client- and server-sides close the connection.

2. Non Functional Requirements

a. User Requirements

The user of the client-side application must have some experience using the graphical user interface window on a desktop, a laptop and Pocket PC devices; using a mouse or stylus to select items from a drop down list; pressing the buttons; and exiting an application by closing its window. Also, the user must possess at least minimal network technology knowledge to understand how the client-server concept works. This includes information such as that the server program must be running first so a client can connect to it. The system will then help the user by providing sufficient information in the Transfer Status Panel and by using message windows. The knowledge as to how a network connected device can be set up must be available.

b. Hardware

The PFTP application does not require any exceptional computational power in both the desktop/laptop and the Pocket PC version. Some minimal system requirements are:

- **PFTP Server:** Pentium III 500 MHz processor,128 MB RAM, Network Interface Card, a static IP address (no DHCP).
- **PFTP Client (desktop/laptop):** Pentium III 500 MHz processor, 128MB RAM, Wireless Network Interface Card
- **PFTP Client (Pocket PC):** Pocket PC ARM compatible handheld, 16MB of RAM, 802.11b wireless adapter

c. Software

The following software must be present so the PFTP application can run properly:

- **PFTP Server:** Windows 98, NT, 2000 or XP operating system and a Java Virtual Machine.
- **PFTP Client (desktop/laptop):** Windows 98, NT, 2000 or XP operating system, Java Virtual Machine and the Java 2 SDK1.4 installed.
- **PFTP Client (Pocket PC):** Pocket PC 2002 or greater and a Java Virtual Machine, such as Jeode JVM, installed.

B. UML DIAGRAMS

1. Use Case Diagrams

The basic actors of the PFTP application are the users of the client-side and the server-side user interface, as well as the client and server systems. Figure 15 demonstrates the possible use cases that the user interface allows:

- The server user adds a file to the server's file database
- The server user removes a file from the server's file database
- The client user asks for a file from the server for the first time
- The client user decides to cancel the file transfer without saving any data already transferred
- The user decides to retrieve a partial transferred file at a later time

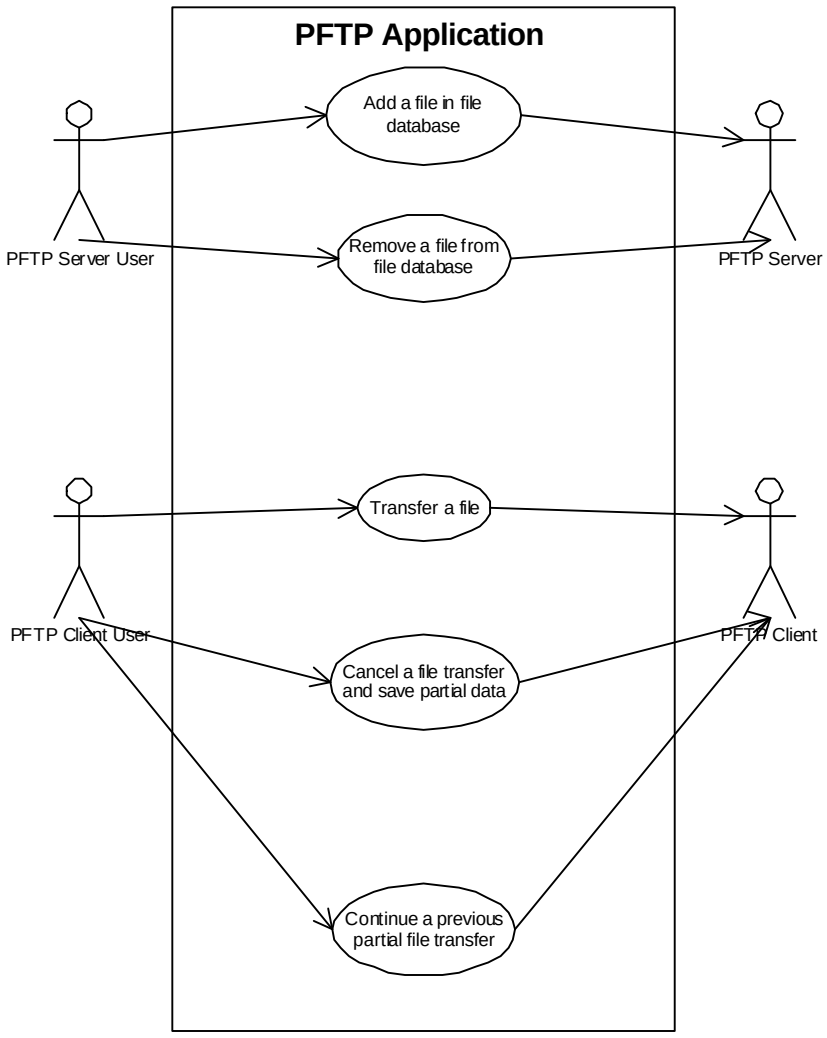

Figure 15. The use case diagram of the PFTP application

2. Class Diagrams

a. User Interface Classes

Figures 16, 17 and 18 show the class diagram of the PFTP client for the desktop/laptop, the Pocket PC and server versions. The user interface for this client-side version is a main frame that contains four panels to control the client and monitor the progress of the file transfer. All the panels except the control panel start by the initiation of a file transfer. After the file is transferred, all panels reset for the next file transfer request.

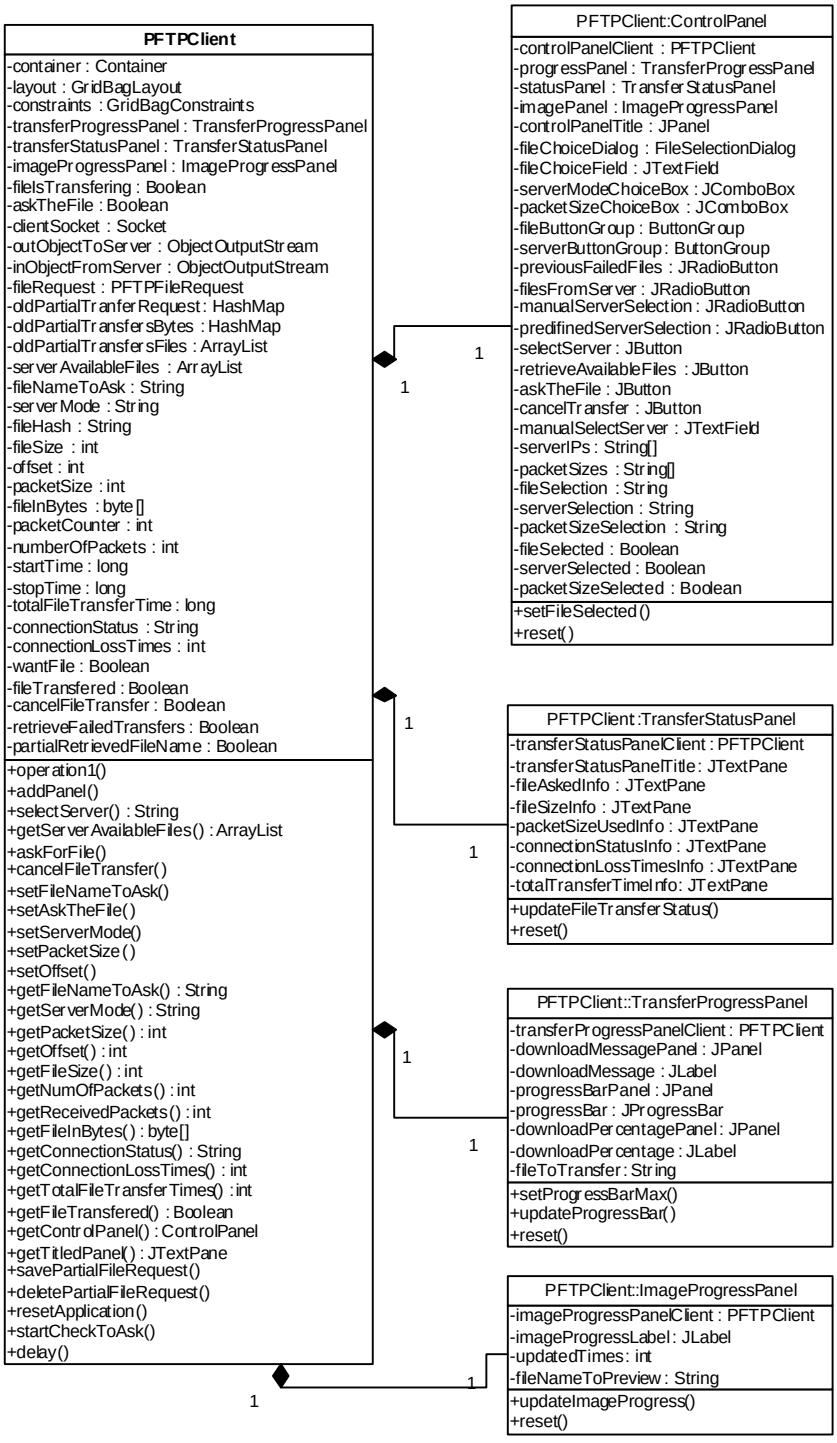

Figure 16. The class diagram of the PFTP Desktop/Laptop Client user interface

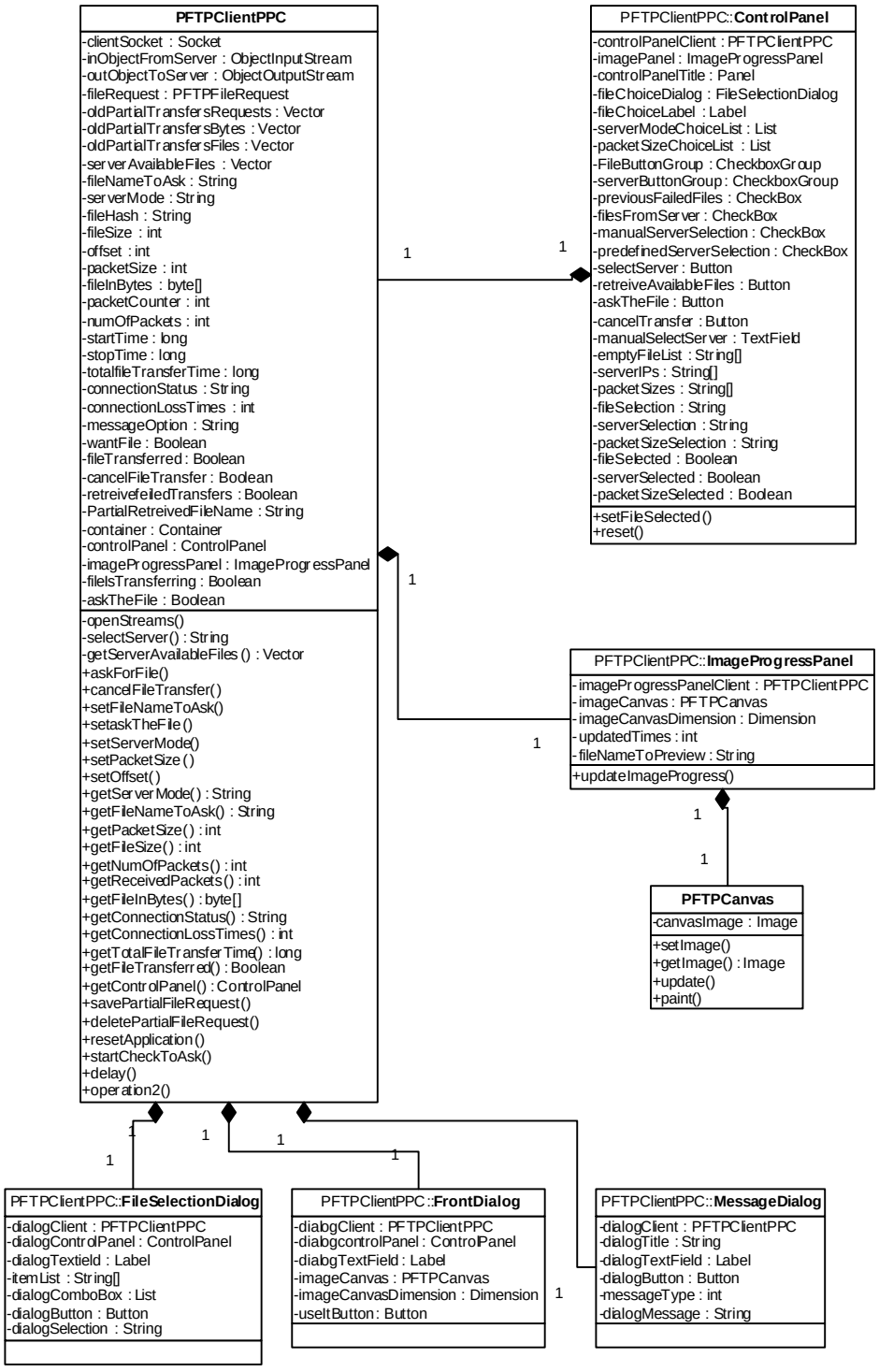

Figure 17. The class diagram of the PFTP Pocket PC Client user interface

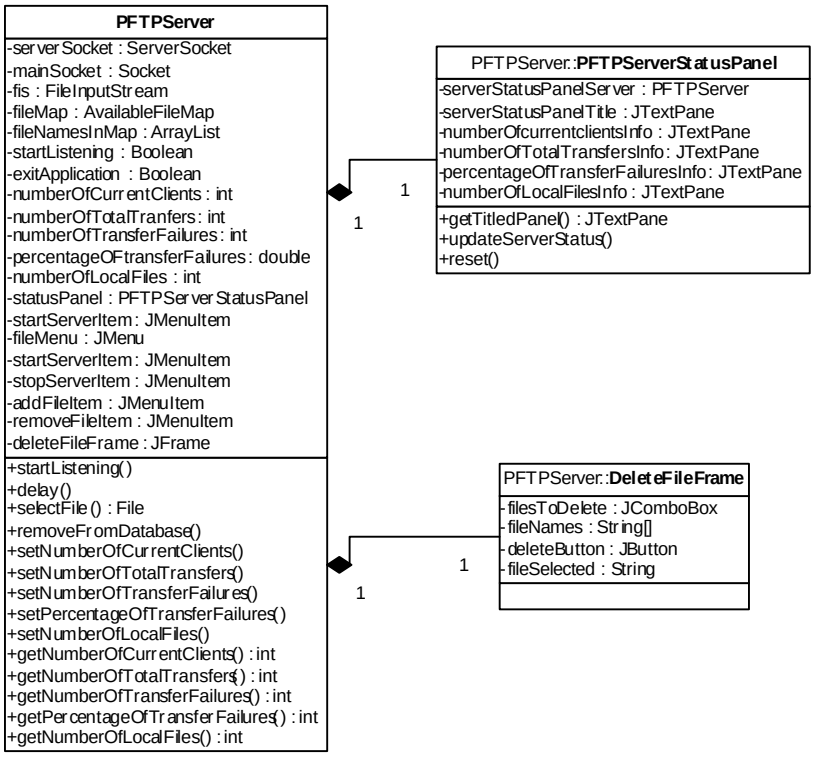

Figure 18. The class diagram of the PFTP Server user interface

b. Desktop/Laptop PFTP Application Version

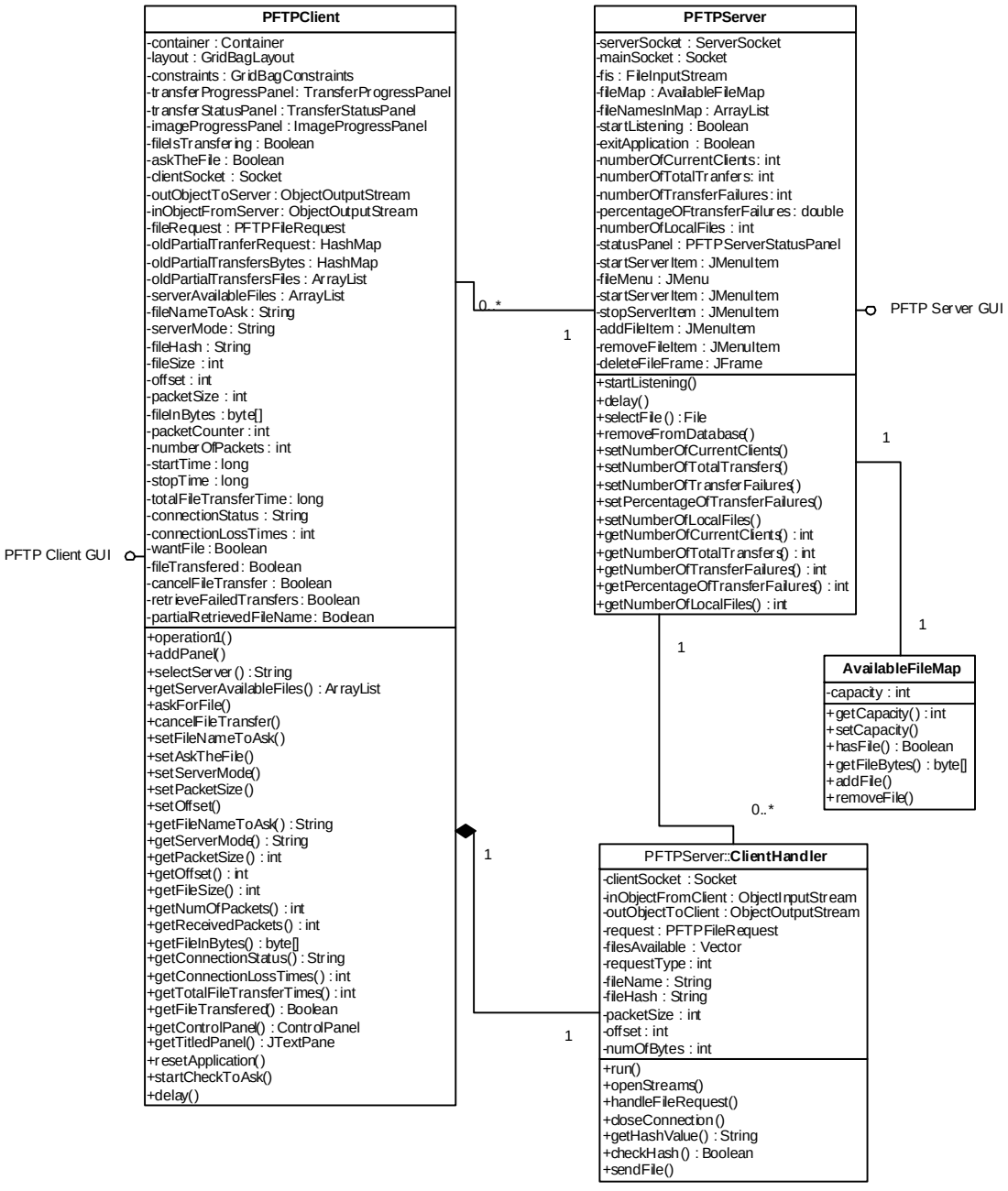

Figure 19. The class diagram of the PFTP application

c. Pocket PC PFTP Application Version

The main difference with desktop/laptop version is that in Pocket PC the classes that were used are less in number because lower version of JDK (1.1.8) is appropriate to use for these devices.

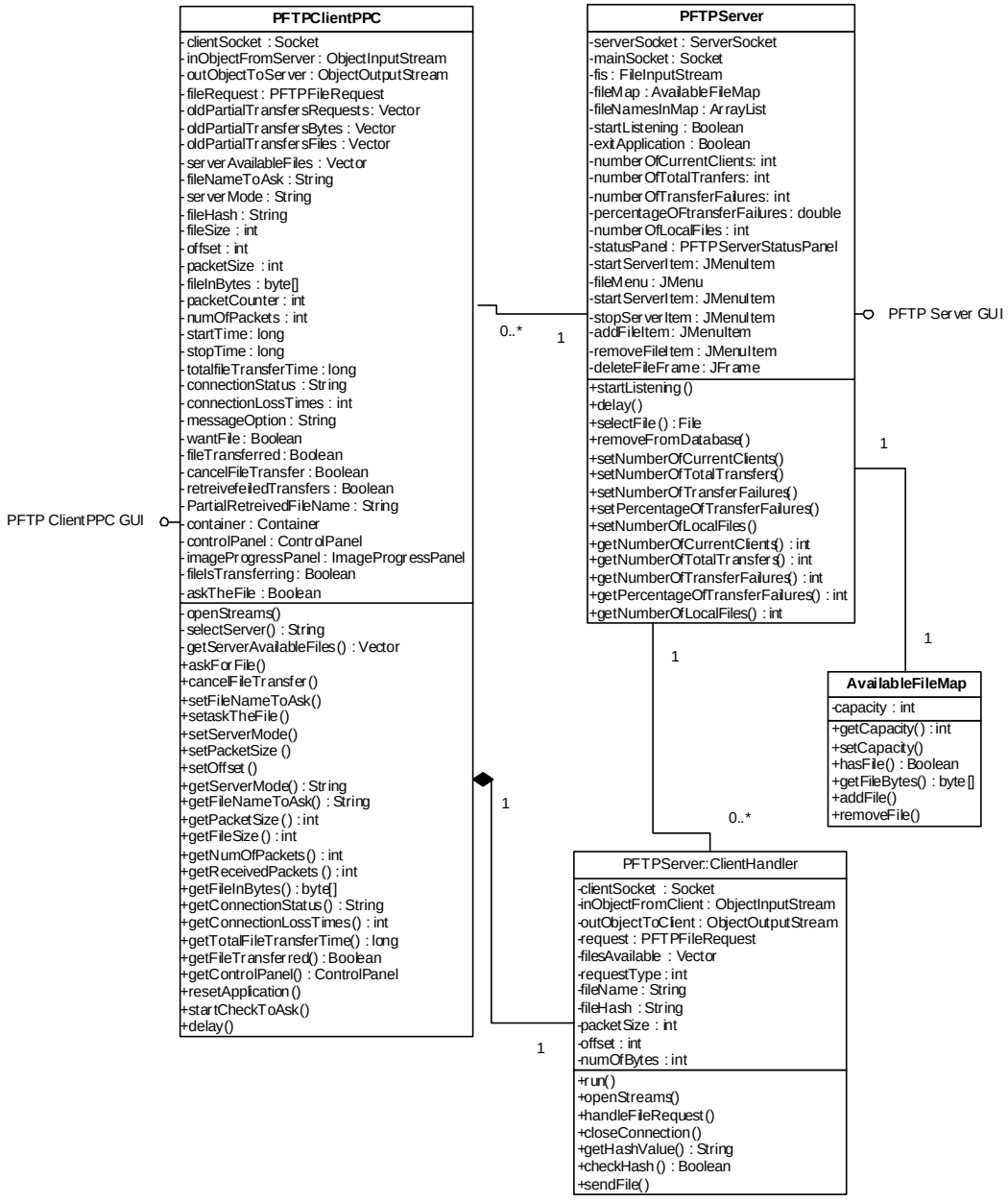

Figure 20. The class diagram of the Pocket PC PFTP Application version

3. Activity Diagram

Figure 21 shows the activity diagram for the PFTP application.

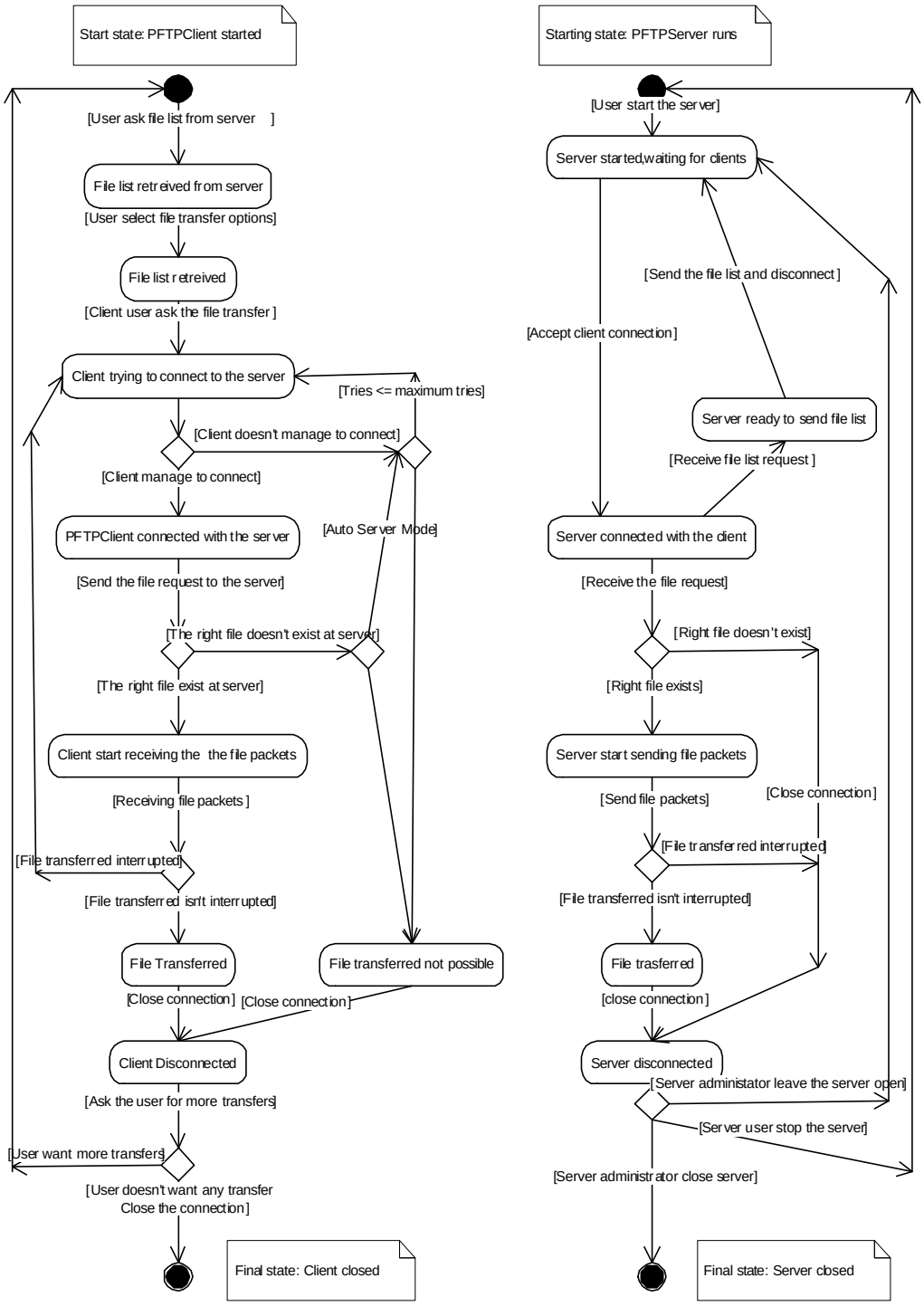

Figure 21. The activity diagram of the PFTP application

4. Flow Sequence Diagrams

The flow sequence diagrams that follow show the order of the interactions between the PFTP application objects during each use-case (mentioned in §B.1) for both versions of PFTP clients (desktop/laptop and Pocket PC).

a. Add a File in Database

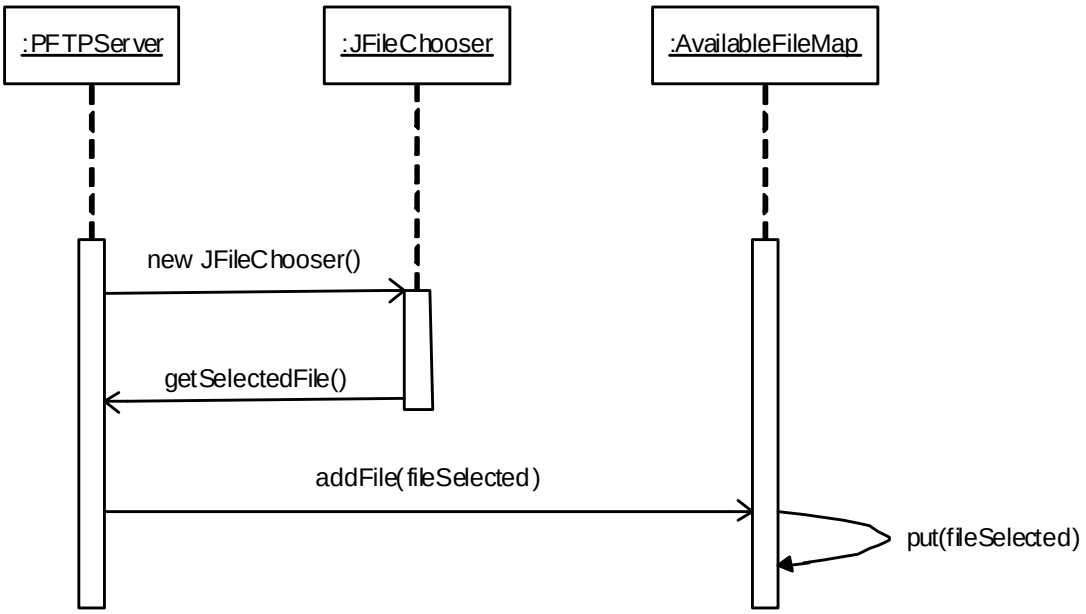

Figure 22. The sequence diagram of “Add a file in database” use-case

b. Remove a File from Database

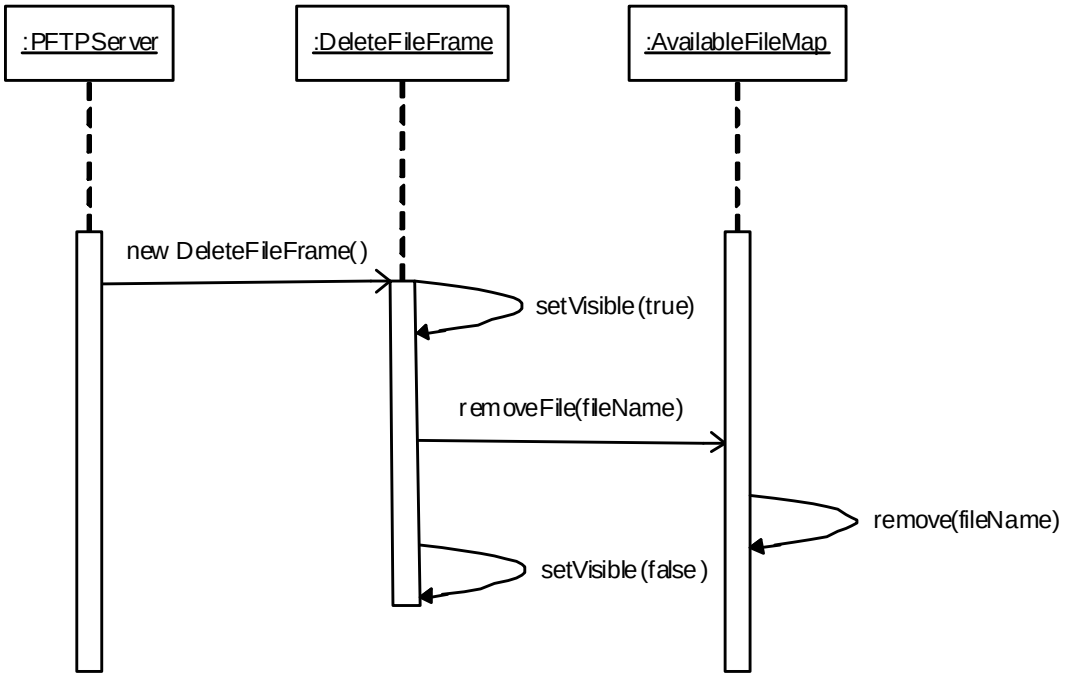

Figure 23. The sequence diagram of “Remove a file in database” use case

c. Transfer a File

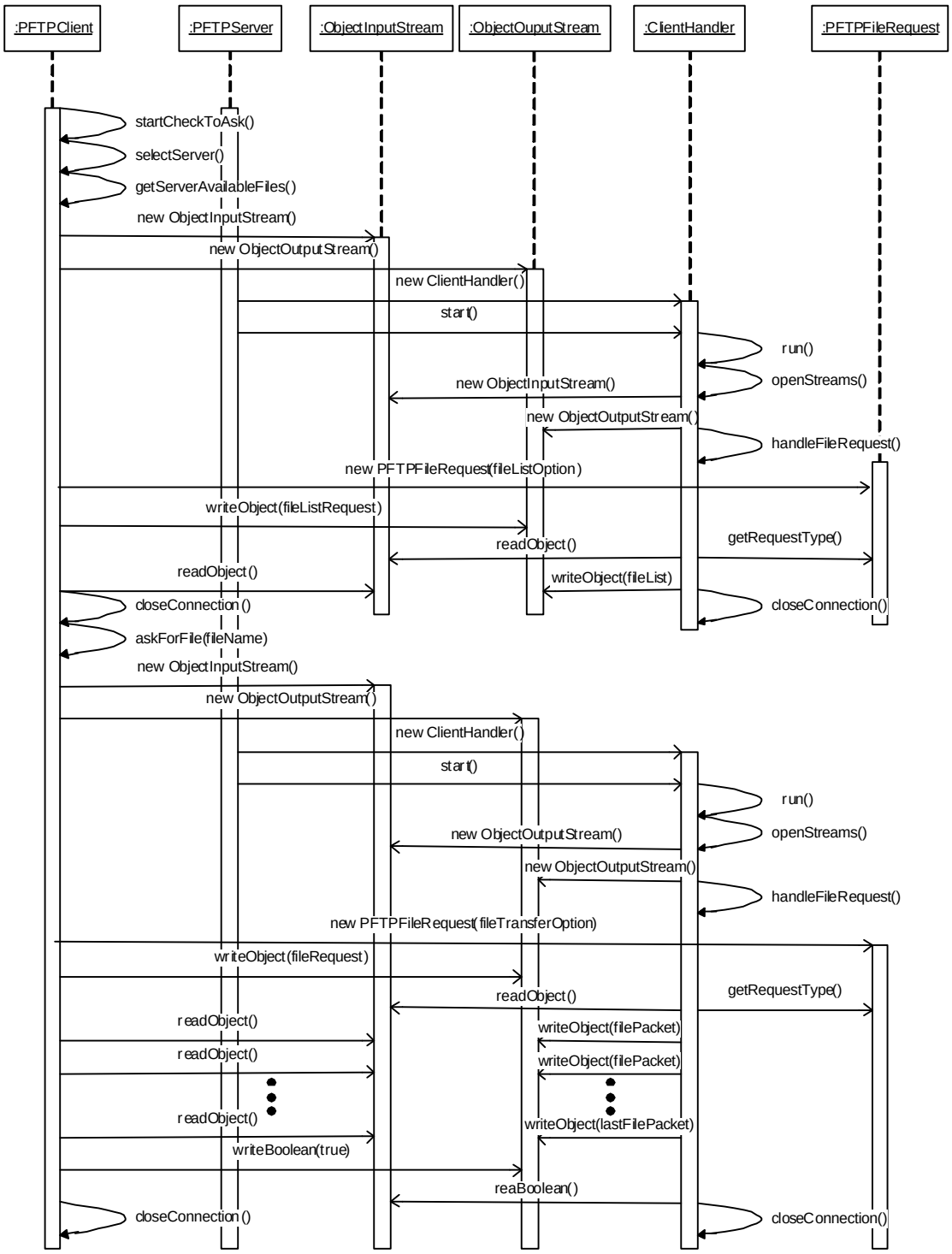

Figure 24. The sequence diagram of “Transfer a file” use-case

d. Cancel File Transfer and Save Partial Data Use Case

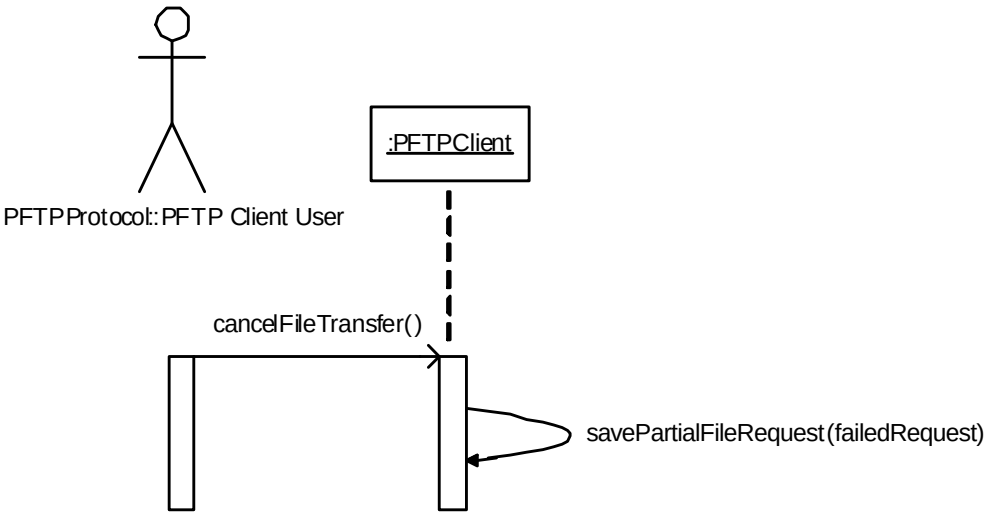

Figure 25. The sequence diagram of “Cancel file transfer and save partial data” use case

e. Continue a Previous Partial File Transfer

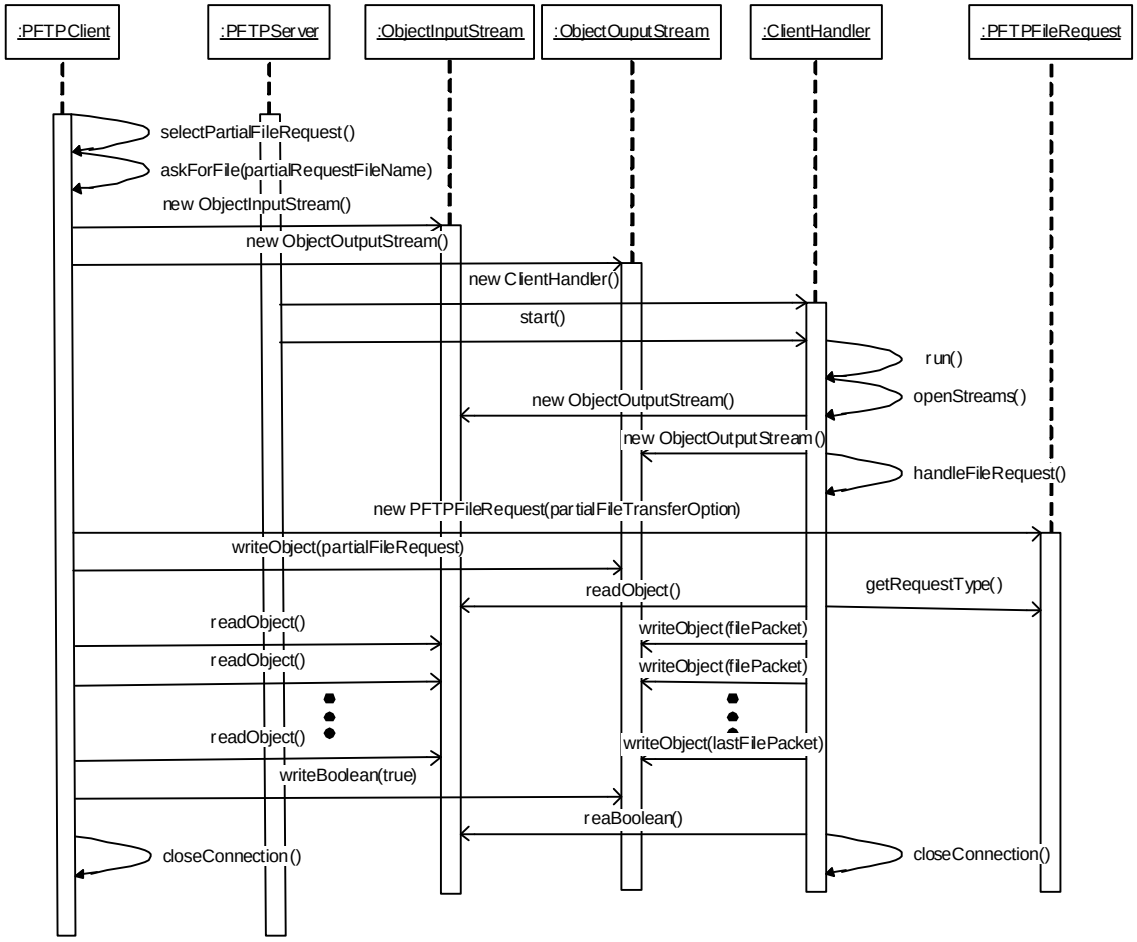

Figure 26. The sequence diagram of “Continue a previous partial file transfer” use case

IV. APPLICATION DEVELOPMENT

This chapter describes the development tools used to build the PFTP application and some development decisions were made during the implementation of some application components such as the user interface and the client-side and the server-side programs.

A. DEVELOPMENT TOOLS

The PFTP application is written using the Java programming language. For both the server-side and the laptop client version, the Java 2 Standard Edition (J2SE) with the Java Development Kit 1.4 (JDK 1.4) was used. In the Pocket PC client version, the Java Development Kit 1.1.8 (JDK 1.1.8) was used because handheld and mobile devices cannot support the functionality provided by greater versions of JDK. The Java code was developed in the JBuilder 9 Java Editor environment.

B. USER INTERFACE

1. PFTP Server

The server side of the application has a limited-functionality user interface so that the user can perform some basic operations such as starting and stopping the PFTP server, and manipulating the database where the files reside for the clients to download. Figure 27 shows the first interface that the user sees when the server program runs.

Figure 27. User interface of the PFTP Server application

There is a menu bar with two menu items. The ***Server*** menu item gives the user the ability to start the server (start listening to the assigned service port), stop the server, or exit the application (Figure 28).

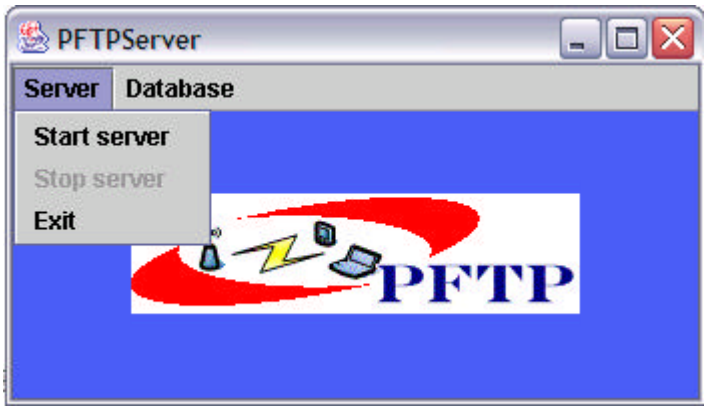

Figure 28. User interface when the **Server** menu item is selected

If the server is started by the user, in the main frame, a server status panel appears that shows how many clients are connected to the server, how many file transfers have occurred since the server started, how many of the file transfers failed, and how many files are available in the database for download (Figure 29).

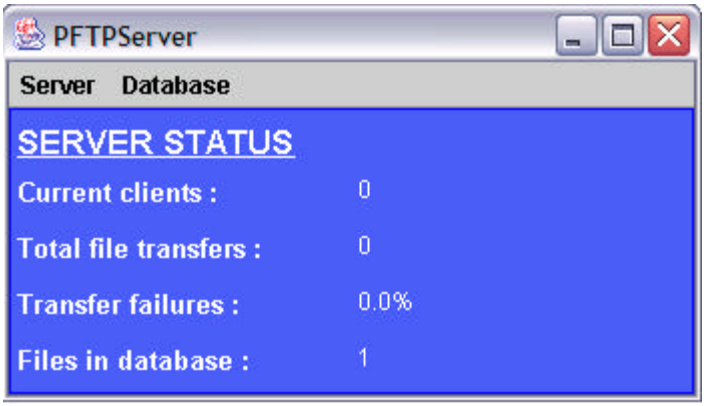

Figure 29. User interface when the server starts

The ***Database*** menu item can be used by the user to update the server file database by adding or removing files (Figure 30). Before the user can start the server, the user must first add files to the database to be available for the client.

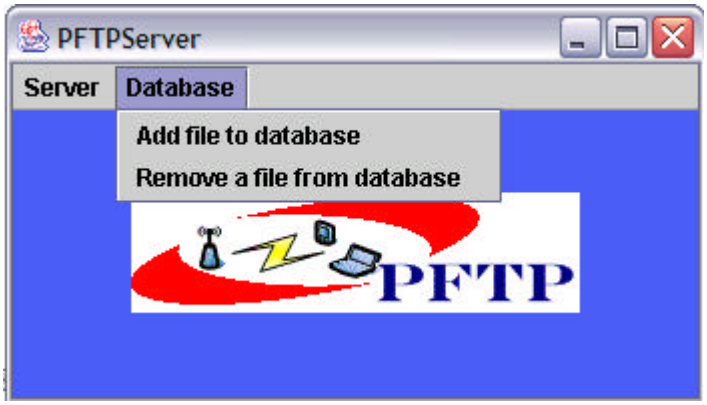

Figure 30. User interface when the **Database** menu item is selected

Selecting the add or remove file option from the Database menu item, the user can browse among the local directories using a popup file dialog (Figure 31) and select the file to add or remove from the server database, respectively. The server's file database does not store the files as a file objects, but rather as individual byte arrays.

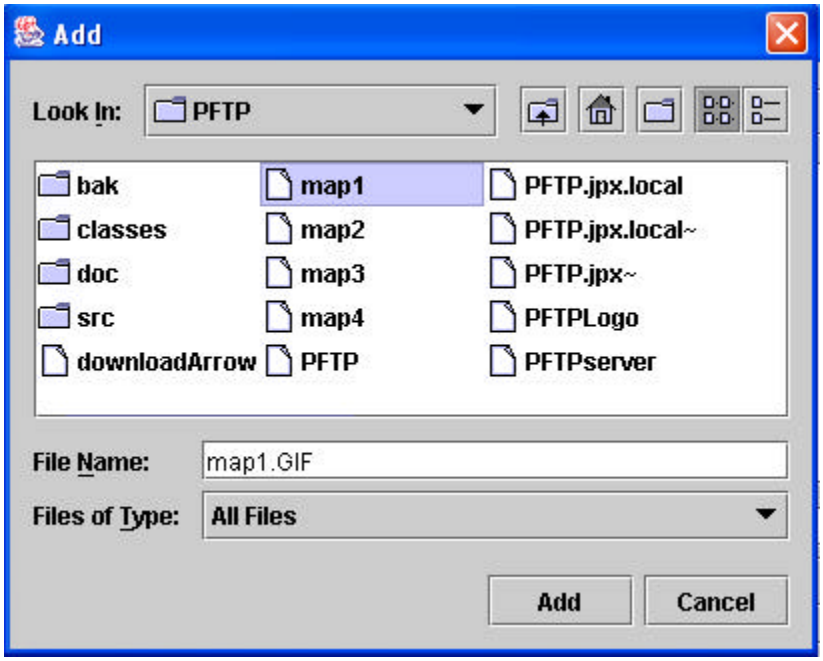

Figure 31. User interface when the **Database** menu item is selected

2. PFTP Client

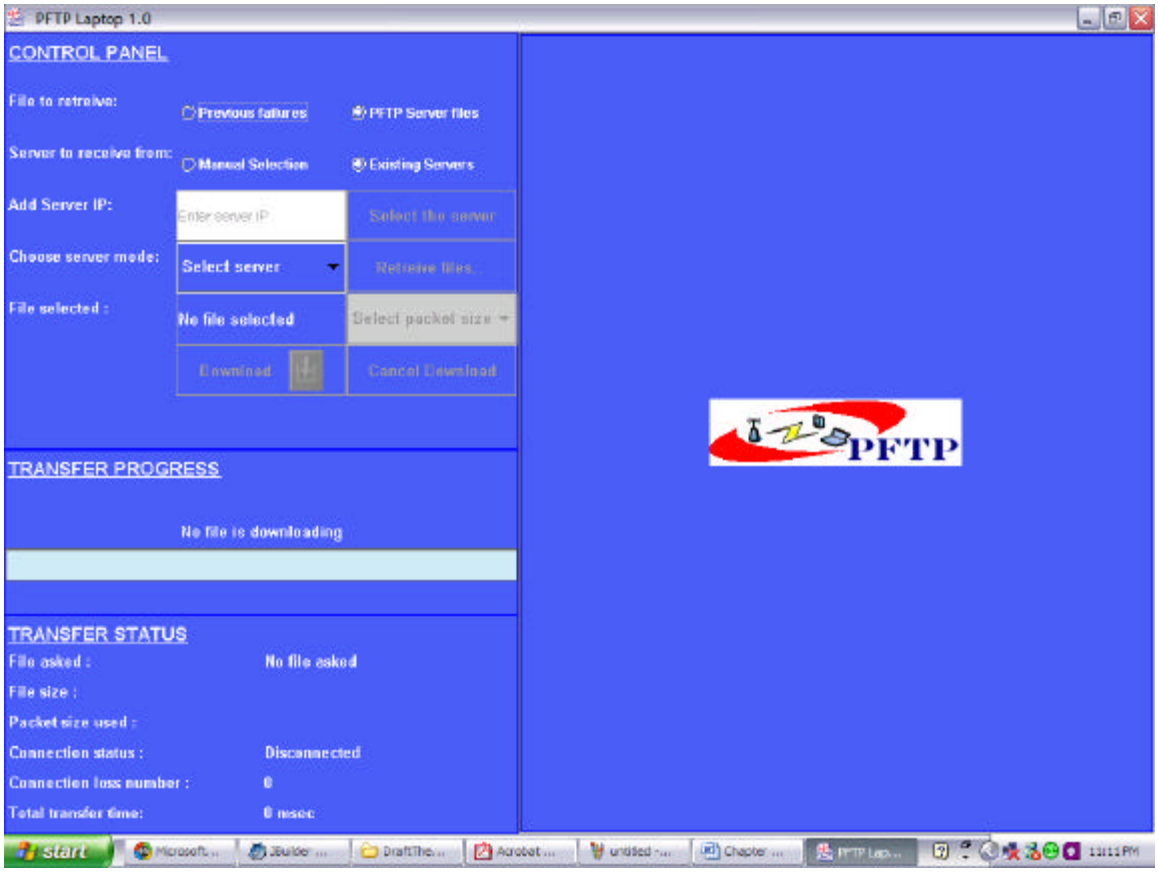

Figure 32. The user interface for the client side in the desktop/laptop version

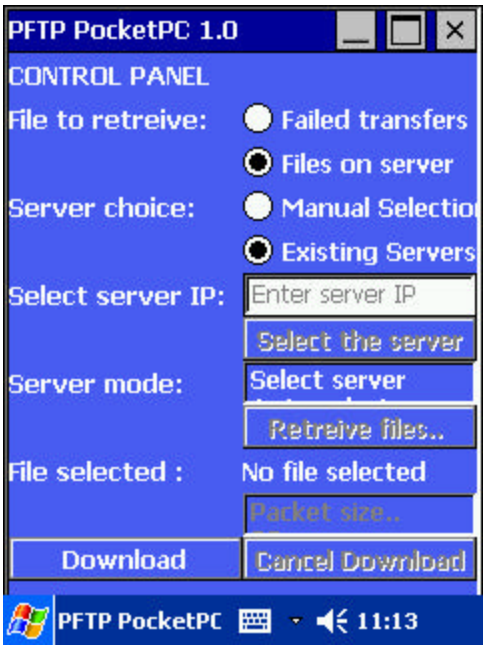

Figure 33. The user interface for the client side in the desktop/laptop version

C. FUNCTIONALITY IMPLEMENTATION

1. File Transfer Failure Recovery

The main functionality of the application is to reconnect the client program when the connection is lost during the file transfer process. The implementation for this feature was based on a loop that the client enters when a file is requested from the server. This loop ensures that the client will keep connecting until it gets the entire file. The control conditions of the loop are:

- The file transfer completion. If the entire file is successfully transferred the client exits the loop.
- The option of the user to cancel the file transfer. If the client user decides to do so, the loop is ended.
- The number of times the client has already tried to recover from the connection loss. If this number is greater than the maximum allowed number of tries, the loop terminates.

2. File Authentication

For the file authentication check performed by the client and the server, the default MD5 algorithm was used to get the file hash value. Then the hash value is converted to a string so that it can be easily exchanged during the communication protocol. Figure 34 shows the code segment getHashValue() used to produce the file hash value.

```
/**
  * Compute the hash value of a given file
  *
  * @param fileName the name of the file
  * @return the String representation of the file hash value
  */
 public String getHashValue( String fileName ){

  String theHash = null;
  try{
    File theFile = new File( fileName );
    fis = new FileInputStream( theFile );
```

```
  byte[] buffer = new byte[8192];
  int length;
  //Select the MD5 hash algorithm
  MessageDigest md = MessageDigest.getInstance( "MD5" );
  while( ( length = fis.read( buffer ) ) != -1 ){
    md.update( buffer, 0, length );
  }
  fis.close(); // Close the file stream to free the file

  // Final computation of the hash
  byte[] hash = md.digest();

  // Create a String version of the hash value using the default
  // 64bit Character encoding algorithm.
  BASE64Encoder encoder = new BASE64Encoder();
  theHash = encoder.encode( hash );

 } catch( FileNotFoundException fnfe ){ // In case file not found
  System.out.println( fnfe );
  theHash = new String( "fileNotFound" );
 } catch( Exception e ){ // In case there is problem
  System.out.println( e ); // with hash computation
  theHash = new String( "hashProblem" );
 }

 return theHash;

} // end getHashValue() method
```

Figure 34. Code segment to produce the file hash value

V. TESTING

This chapter describes to the testing process of the PFTP application, including the test network description, the various scenarios that were used, and the general results of the testing.

A. TESTING NETWORK DESCRIPTION

The testing of the PFTP application required the installation of a basic wireless network that simulates a real situation in which all the application components' operations can be tested against several scenarios.

1. Considerations-Limitations

- The NPS wireless network was used to test the mobile PFTP clients
- The IP addresses of the PFTP server must be static.
- The number of wired servers and wireless enabled devices used to build the test network, were sufficient to test the requirements of the thesis research and not to test the volume of client traffic that the application can handle
- In the most of the testing scenarios, a "slower" version of the PFTP is used. That means the server's and client's program response was slowed with the use of a "delay" function so that it is easier for the user to observe the communication protocol features and behavior during the test scenarios.
- The desired connectivity failures necessary to test the protocol responses were manually caused by either unplugging the network connection in the wired devices (mainly servers), or by disabling or removing the wireless adapters from the wireless enabled mobile client devices (laptop, PocketPC).

2. Testing Network

As Figure 35 shows, the testing network consists of three PFTP servers, each a wired device, part of the NPS network, one or both wireless enabled laptop and Pocket PC devices running the appropriate PFTP Client version software.

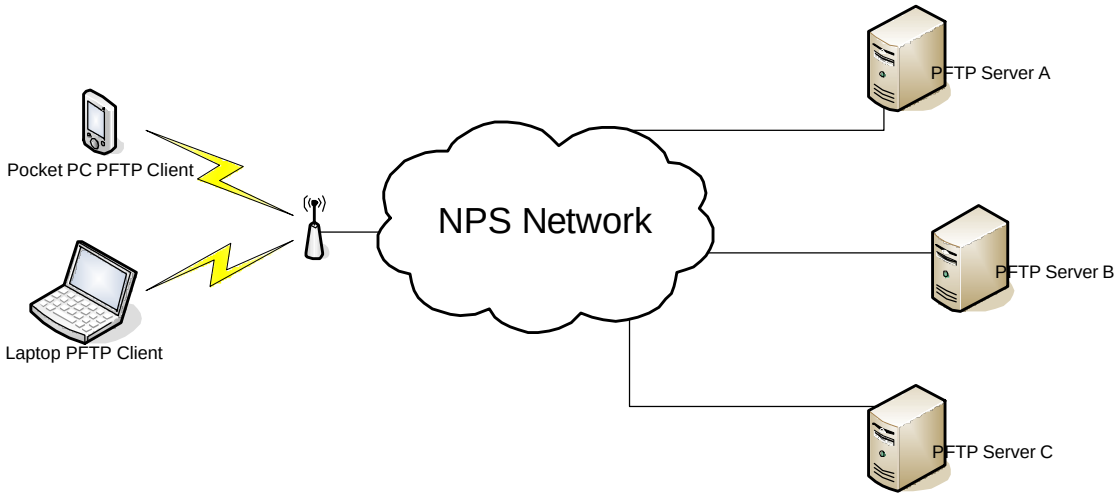

Figure 35. The testing network of PFTP application

B. TESTING SCENARIOS

Several testing were developed to emulate the various problems that might be encountered by a wireless device while downloading a large file. These scenarios assured that the purposes of the thesis research were fulfilled. Table 2 lists all the testing scenarios associating them with a reference code so that they can be referred to the results description without naming them explicitly. The reference codes start with a group of letters that indicates the general scenario type and ends with a counter number. The reference code part that refers to the scenario's type is one of the following:

- SUIS: Server User Interface Scenario. Situations that can happen during the interaction of the PFTP Server's user with the available user interface.
- CUIS: Client User Interface Scenario. Situations that can happen during the interaction of the PFTP Client's user with the available user interface.
- CPS: Communication Protocol Scenario. Situations that can happen during the communication between the client and the server.

Scenario Reference Code	Scenario's description
SUIS-1	Starting the server without adding files in the database
SUIS-2	Trying to remove a file from the database when it is empty
SUIS-3	Entering the same file twice into the database
CUIS-1	Selecting to continue a previously failed file transfers when one is not an associated file saved
CUIS-2	Trying to download a file without selecting all the necessary file transfer options
CPS-1	The PFTP server is not reachable when the available files are requested and the single server mode is selected
CPS-2	The PFTP server is not reachable when the available files are requested and the auto server mode is selected
CPS-3	During the file transfer, the connection is lost but is restored again after a short period of time. The single server mode is selected.
CPS-4	During the file transfer, the connection is lost but is restored again after a short period of time. The auto server mode is selected.
CPS-5	During the file transfer, the connection is lost but is restored again after some period of time. The auto server mode is selected and all the other servers except the first one selected, have the same named file but contain different content.

Table 2. The testing scenarios for the PFTP application

If the server has been started by the user, the main frame of the server's status panel appears in that shows how many clients are connected to the server, how many file transfers have occurred since the server started, how many of the file transfers failed, and how many files are available in the database for download (Figure 3).

C. TESTING RESULTS

The following paragraphs explain how the network components responded to each of the scenarios and how the user interface helped the user be informed if a file transfer failed during its operation. The reference codes listed in Table 1 are used to refer to each scenario.

- **SUIS-1**. The scenario tries to determine if the PFTP server starts listening to the assigned port without having any files available for the clients to download. The result was that the program informed the user via a message to add at least one file to the server's file database such that the server has a reason to start listening for clients (Figure 2).

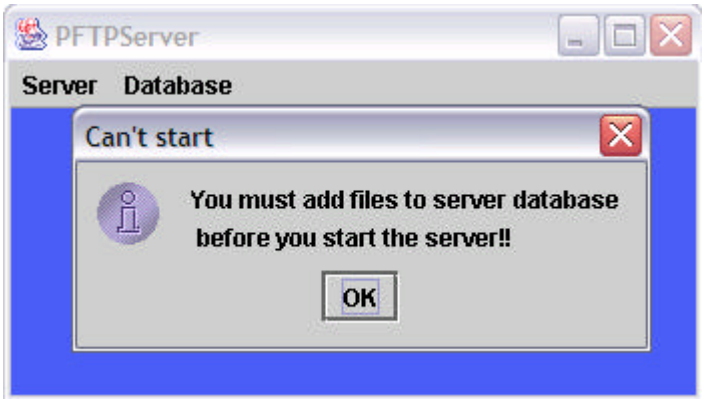

Figure 36. The response of server side in the SUIS-1 scenario

- **SUIS-2**. The same situation as the previous but this time tested function is an attempt to remove a file from the empty database. When the database is empty, the selection of the user to remove a file caused a message to appear informing the user about the database's status (Figure 37).

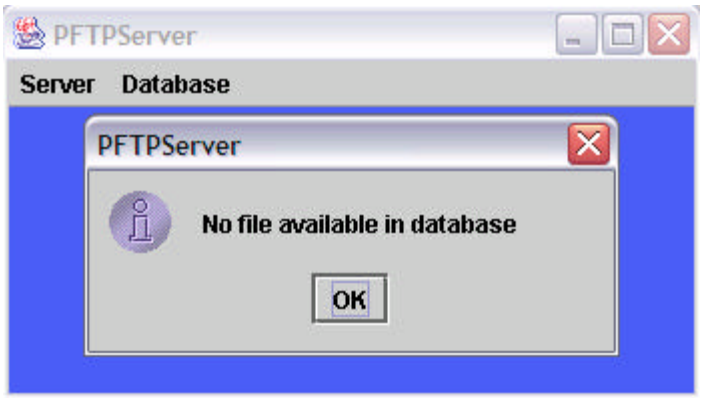

Figure 37. The response of server side in the SUIS-2 scenario

- **SUIS-3**. Trying to add a file to the file database that has the same filename as an existing one, is another tested case. In this scenario, the response of the server program was to ask the user to choose between replacing the existing file with the new one or leaving the old file in the database (Figure 38).

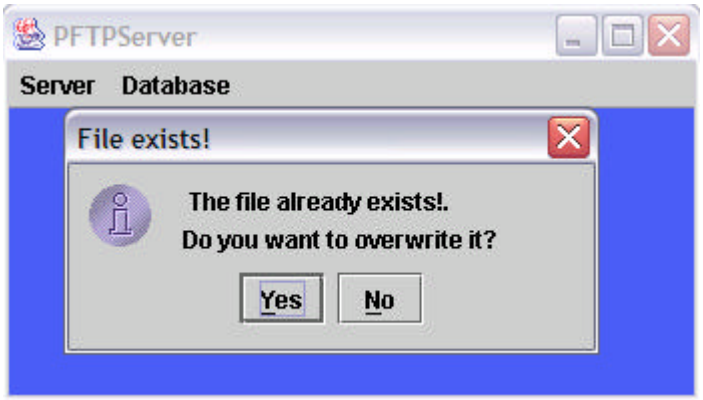

Figure 38. The message when a already existing file is added in file database

- **CUIS-1**. The user interface gives the user the ability to select to continue a previously failed file transfer, saving the partial data to a form in the database. However, when no failed transfers have occurred, the selection of the previous file retrieval mode causes the system to respond with a message that states this (Figures 39 and 40).

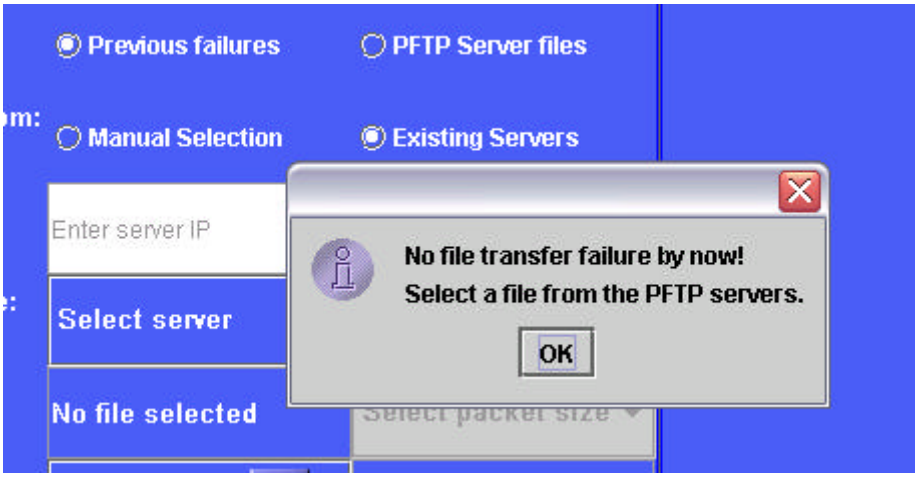

Figure 39. The message when no previous partial file transfer exists

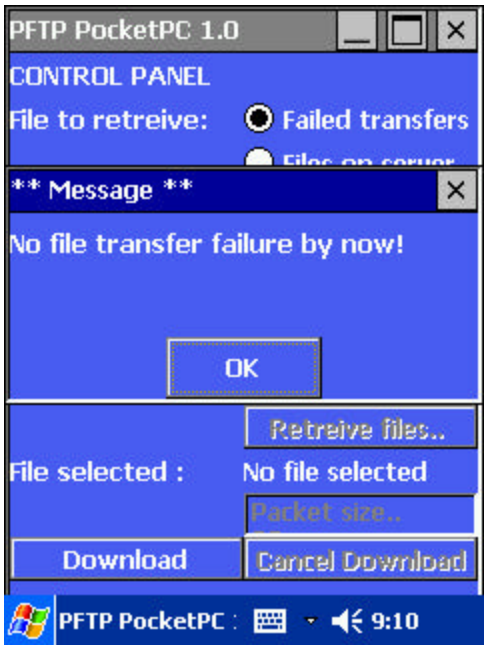

Figure 40. The same message as Figure 5 in Pocket PC device

- **CUIS-2**. In this scenario, the user presses the download button, without first selecting all the transfer options such as the file name, the server, or the packet size. The system informs the user to do so (Figures 41 and 42).

Figure 41. The message when not all the file transfer option are selected

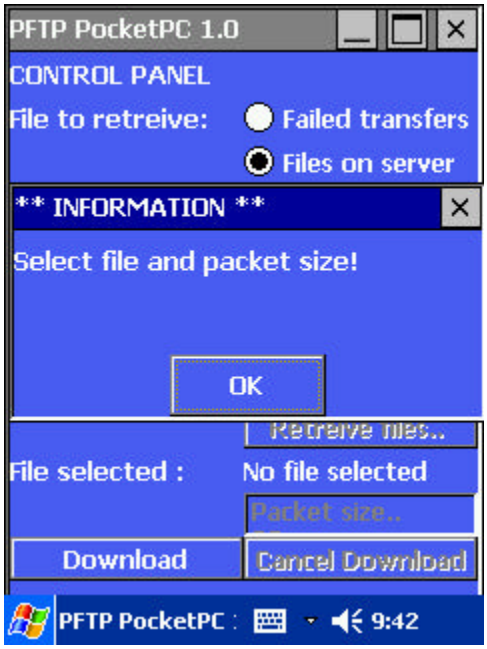

Figure 42. The same message as Figure 7 in Pocket PC environment

- **CPS-3**. This is the main function of the PFTP application-recovering from a failed connection. During the file transfer, the connection is interrupted and the client attempts to the server in either single and or manually selected server mode. The image progress panel of the user interface, which displays the image download progress, stops updating until the client reconnects to the server and begins retrieval of the rest of the file. Figures 43 and 44 show the paused state of the user interface during connection failure and Figure 45 the attempt of the client program to connect to the server. Figures 46 and 47 show the message when the file is finally transferred, followed by the amount of time the transfer required.

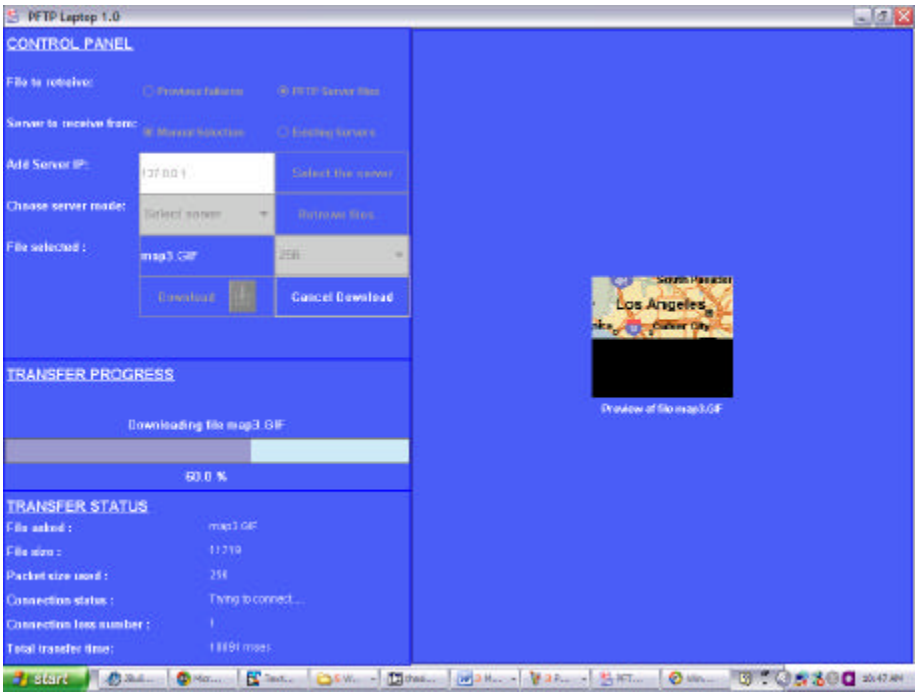

Figure 43. Paused state of the user interface after a connection loss (laptop)

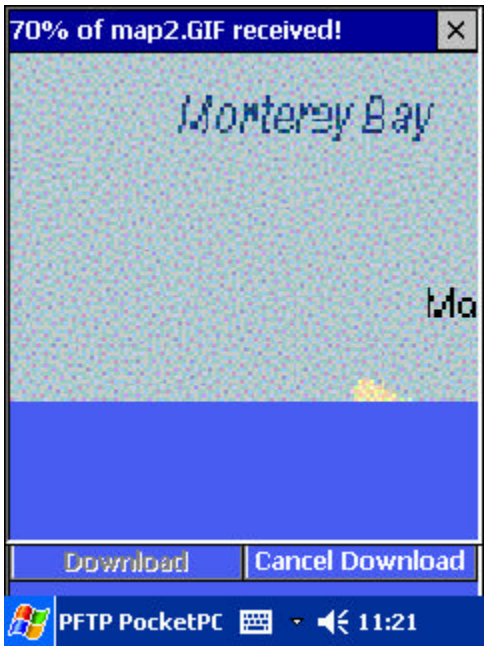

Figure 44. Paused state of the user interface after a connection loss (Pocket PC)

```
Bytes 8193 to 8448 received
Bytes 8449 to 8704 received
Problem reading I/O Streams
Transfer cancelled!
Trying to connect to the server....169.254.1.190
Transfer cancelled!
Trying to connect to the server....169.254.1.190
Transfer cancelled!
Trying to connect to the server....169.254.1.190
Transfer cancelled!
Trying to connect to the server....169.254.1.190
Transfer cancelled!
Trying to connect to the server....169.254.1.190
Transfer cancelled!
Trying to connect to the server....169.254.1.190
Client connected!
File size to receive 11719 bytes
Bytes 8705 to 8960 received
Bytes 8961 to 9216 received
```

Figure 45. The system output that shows the reconnection procedure

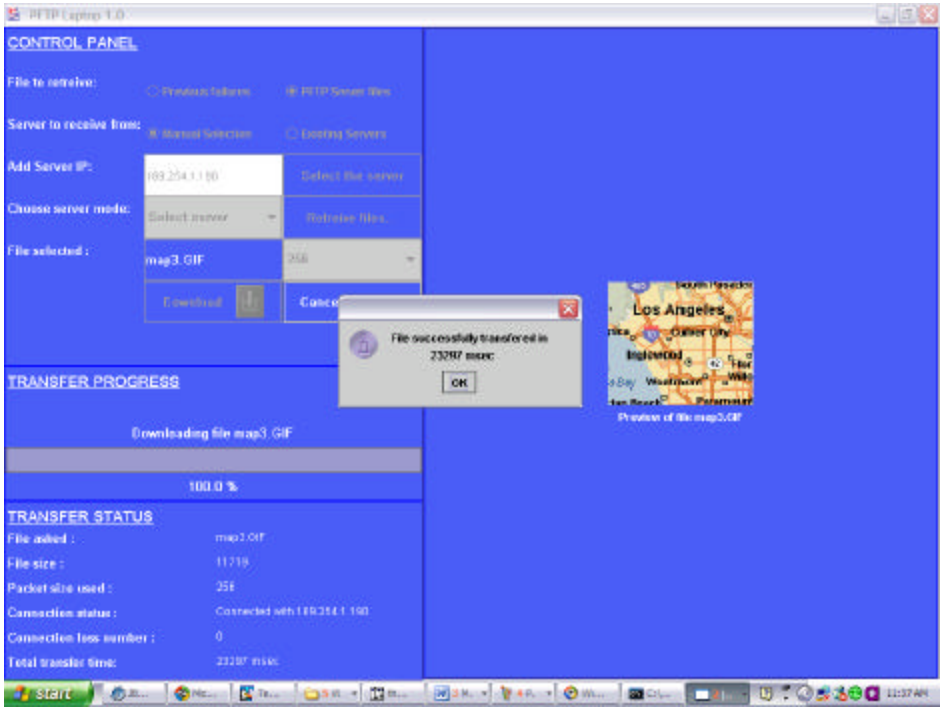

Figure 46. The user interface when the file is transfer is completed in the laptop version

Figure 47. The user interface when the file transfer is completed in the Pocket PC version

- **CPS-4**. The difference between this scenario and that of CPS-3 is that the server mode is auto mode. When the connection is lost, the client randomly picks a server randomly from the valid server list and tries to that server. Figure 48 shows the system output of the client trying to connect to the valid servers. Hopefully, it finds a server to continue the partial file transfer.

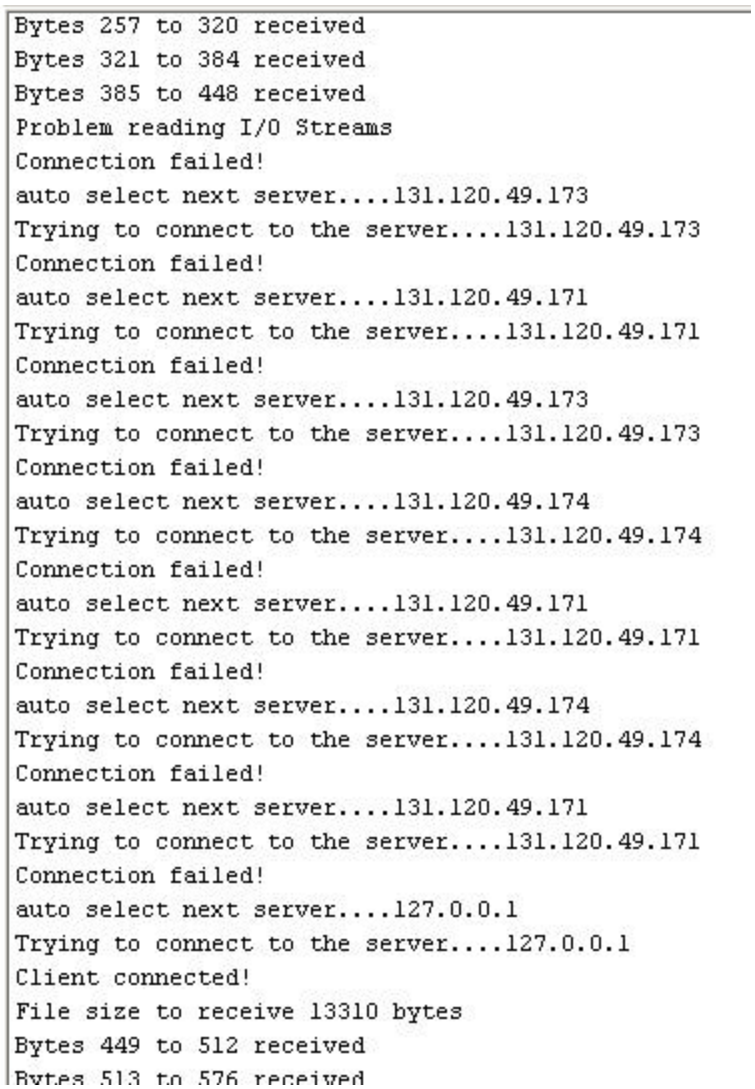

Figure 48. Auto server search after a connection loss in auto server mode

- **CPS-5**. This last scenario tested the response of the client when, after a connection loss, it finds a server that has a file with the same name but a different file hash value. In that case, the server sent a response that informed the client that the file version in its database is not the requested one.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. CONCLUSIONS AND FUTURE WORK

A. SUMMARY

The potential goal of this thesis research was to model an upper-layer communication protocol that can preserve file transfer sessions in a highly mobile environment, and implement an application that use and demonstrates the operation of this protocol. As a background research, all the well-known file transfer protocols were examined in aspect of their communication protocol characteristics and if there is any feature that can be used to achieve a partial file transfer after a connection failure.

The communication protocol that was designed is a client-server-type protocol that supports client multithreading in server-side and can dynamically recover from a file transfer failure. This partial file recovery feature is achieved by designing and implement the client-side program to keep attempting several times asking the server for the part of the file was not received when a connection failure occurs.

Also, the PFTP application was developed to use this protocol and make user able to interact with protocol and the file transfer features. User interface was designed so that the dynamic partial file retrieval and the can be visible to the use at real time. In order to make this function possible, special panels created on the user interface that show the file transfer progress and displays the partial file data received

The communication protocol and the PFTP application tested against several scenarios. The main situation that the protocol behavior was tested was when during the file transfer the connection failed that caused the client program retry successfully reconnecting when in a reasonable time the connection was restored. PFTP application also tested in the file management on the server and the multithreaded behavior. The client side in both laptop and Pocket PC versions tested successfully in visualizing the file transfer progress and in controlling the file transfer options (file request, canceling the downloading, or choosing to continue previous failed file transfers).

B. FURTHER WORK

Extending the research scope of this thesis and the application developed in support of it, there are issues that raise opportunities for further research. These include:

1. Communication Protocol Design

Using as a model the PFTP client-server communication protocol, more research could be focused on enhancing specific aspect of the protocol it self and how its operation can be improved. Some key areas towards this direction could be:

- Multiple types of session content. Further enhancement in the protocol can be done, so various types of sessions (e.g. steaming media, interactive connection, etc.) can be supported instead of just file transfers.
- Use of UDP protocol. The PFTP communication model uses TCP as the underlying protocol in the data transfer connection. It would be useful implementing the same application using the UDP protocol, and comparing the performance of the two implementations with respect to data session survivability in a wireless environment.
- PFTP server locator. Additional feature of the application develop a content/server location protocol that will automatic discover server to re-establish sessions instead of current manual insertion of servers.

2. Application Development

In application development the following further work can be done:

- Extension the API class library. Create a class or library API for the persistent connection protocol that application developers can use to develop new applications.
- Use persistent data sessions. Examine applying the persistent data sessions in other type of application where can be useful.

APPENDIX. CLASS SOURCE CODE

```
//packages for network components
import java.io.*;
import java.net.*;

//packages for user interface components
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import javax.swing.*;
import javax.swing.text.*;
import javax.swing.SwingUtilities;

//package for security components
import java.security.*;

//package for character encoding components
import sun.misc.*;

/**
 * File PFTPClient.java
 *
 * This is a FTP Client of the application
 *
 * This client ask for a file and stays in a loop keep asking
 * until it gets it all. In the first request it receives the
 * hash value of the requested file so when the connection is
 * lost client can ask an other server the same file with out
 * worrying about the authenticity of it.
 *
 * @author LT Periklis Pantoleon
 * @version 1.0
 */
class PFTPClient extends JFrame{

//----------------------------------------
//
//    Private Data Members:
//
//----------------------------------------

  /** The default port where the FTP client is asking the server */
  private static final int FTP_PORT = 6789;
```

```
/** The string array with the valid servers can be chosen  */
private static final String[] VALID_SERVERS = {
   "131.120.105.237", "131.120.105.250", "131.120.201.46", "127.0.0.1"};

/** The packet size array with the packet sizes can be chosen  */
private String[] PACKET_SIZES = {new String( "32" ),
   new String( "64" ), new String( "128" ),
   new String( "256" ), new String( "512" ),
   new String( "1024" )};

/** The connected status */
private static final String CONNECTED = "Connected";

/** The disconnected status */
private static final String DISCONNECTED = "Disconnected";

/** The status when the client is trying to connect */
private static final String TRYING_TO_CONNECT = "Trying to connect.....";

/** The timeout period of the TCP sockets  */
private static final int SO_TIME_OUT = 3000;

/** The length in bytes of the file hash value */
private static final int MD5_HASH_LENGTH = 16;

/** The font size for the panels' titles */
private static final int PANEL_TITLE_FONT_SIZE = 17;

/** The font size for the panels' labels */
private static final int PANEL_LABEL_FONT_SIZE = 13;

/** The panels' font color */
private static final Color FONT_COLOR = Color.white;

/** The color of the background of the frames and panels */
private static final Color BACKGROUND_COLOR = new Color( 77, 92, 240 );

/** The color of the panels' border */
private static final Color PANEL_BORDER_COLOR = Color.blue;

/** The maximum reconnection tries each time client lose connection*/
private static final int MAX_RECONNECTION_TRIES = 3;

/** The maximum reconnection tries each time client lose connection */
private static final int MAX_TOTAL_TRIES = 3;
```

```
/** The times that the image is updated in the image progress panel */
private static final int IMAGE_UPDATE_TIMES = 10;

/** The request type when a file is asked */
private static final int FILE_REQUEST_TYPE = 1;

/** The request type when a file array is asked */
private static final int FILE_ARRAY_REQUEST_TYPE = 0;

/** The file name of the logo image of the client frame */
private static final String LOGO_IMAGE = "pftpLogoClient.gif";

//-------------------------------------------------
//
//    Data Members for the FTP Client functionality:
//
//-------------------------------------------------

/** The socket client use to communicate with the server  */
private Socket clientSocket;

/** The object input stream to receive objects from client */
private ObjectInputStream inObjectFromServer;

/** The object output stream to send objects to client */
private ObjectOutputStream outObjectToServer;

/** The file request received from client */
private PFTPFileRequest fileRequest;

/** The map that store that old partial file bytes */
private HashMap oldPartialTransferRequests;

/** The map that store that old partial file bytes */
private HashMap oldPartialTransfersBytes;

/** The array of the file names that failed to downloaded */
private ArrayList oldPartialTransfersFiles;

/* The array list of the files available on PFTP server */
private ArrayList serverAvailableFiles;

/** The name of the asked file */
private String fileNameToAsk;

/** The server mode selected */
```

```
private String serverMode;

/** The server selected */
private String serverSelected;

/** The hash value of the file asked */
private String fileHash;

/** The size in bytes of the file asked */
private int fileSize;

/** The offset from where we need the file */
private int offset;

/** The packet size used for the file transfer */
private int packetSize;

/** The file as a byte array */
private byte[] fileInBytes;

/** The packet counter */
private int packetCounter;

/** The number of packets used for the file*/
private int numOfPackets;

/** The start time of the file reception */
private long startTime;

/** The end time of the file reception */
private long stopTime;

/** The total file transfer time */
private long totalFileTransferTime;

/** The connection status */
private String connectionStatus;

/** The connection loss times */
private int connectionLossTimes;

/** The boolean value of the intetion of the user to ask a file */
private boolean wantFile;

/** The boolean value if the file is transfered or not */
private boolean fileTransfered;
```

```
  /** The boolean value the file transfer is canceled or not */
  private boolean cancelFileTransfer;

  /** The boolean value if a file selected or not  */
  private boolean retrieveFailedTransfers;

  /** The boolean value of the intention of the user to ask a file */
  private String partialRetreivedFileName;

  //-------------------------------------------------
  //
  //    Data Members for the User Interface:
  //
  //-------------------------------------------------

  /** The container of the main frame */
  private Container container;

  /** The layout manager of the container */
  private GridBagLayout layout;

  /** The GridBagConstraints variable of the layout manager*/
  private GridBagConstraints constraints;

  /** The control panel */
  private ControlPanel controlPanel;

  /** The panel that shows the transfer progress */
  private TransferProgressPanel transferProgressPanel;

  /** The panel that shows the transfer status during downloading */
  private TransferStatusPanel transferStatusPanel;

  /** The panel that displays the image during the downloading */
  private ImageProgressPanel imageProgressPanel;

  /** The boolean tranfering status */
  private boolean fileIsTransfering;

  /** The boolean value if user want to download the file or not */
  private boolean askTheFile;

//----------------------------------------
//
//    Constructor:
```

```
//
//---------------------------------------

  /**
   * Default constructor
   */
  public PFTPClient(){

    super( "PFTP Laptop 1.0" );

    // get content pane
    container = getContentPane();

    // instantiate gridbag constraints
    constraints = new GridBagConstraints();

    // create the panels
    transferProgressPanel = new TransferProgressPanel( this );
    transferStatusPanel = new TransferStatusPanel( this );
    imageProgressPanel = new ImageProgressPanel( this );
    controlPanel = new ControlPanel( this, transferProgressPanel,
                                transferStatusPanel,
                                imageProgressPanel );

    // set the layout of the container
    layout = new GridBagLayout();
    container.setLayout( layout );

    // add the panels on the main frame's panels
    //constraints.weightx = 1;
    //constraints.weighty = 1;
    constraints.fill = GridBagConstraints.BOTH;
    addPanel( controlPanel, 0, 0, 1, 4 );

    //constraints.weightx = 1;
    //constraints.weighty = 1;
    constraints.fill = GridBagConstraints.BOTH;
    addPanel( transferProgressPanel, 4, 0, 1, 1 );

    //constraints.weightx = 1;
    //constraints.weighty = 1;
    constraints.fill = GridBagConstraints.BOTH;
    addPanel( transferStatusPanel, 5, 0, 1, 3 );

    constraints.weightx = 1;
    constraints.weighty = 1;
```

```
  constraints.fill = GridBagConstraints.BOTH;
  addPanel( new JScrollPane( imageProgressPanel ), 0, 1, 1, 8 );

  //sets other attributes of the main frame
  setSize( 700, 500 );
  setLocation( 50, 100 );
  setVisible( true );
  setExtendedState( JFrame.MAXIMIZED_BOTH );
  setResizable( true );

  setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );

  //initialization of the private members
  oldPartialTransferRequests = new HashMap();
  oldPartialTransfersBytes = new HashMap();
  oldPartialTransfersFiles = new ArrayList();
  serverAvailableFiles = new ArrayList();

  fileInBytes = null;
  fileNameToAsk = null;
  fileSize = 0;
  packetSize = 0;
  fileHash = "";
  connectionStatus = DISCONNECTED;
  connectionLossTimes = 0;
  totalFileTransferTime = 0;
  retrieveFailedTransfers = false;
  fileIsTransfering = false;
  askTheFile = false;

} // end default constructor

/**
 * Add a component using the GridBagLayout
 *
 * @param component the component to add
 * @param row       the row to add the component
 * @param column    the column to add the component
 * @param width     the width in columns of the component
 * @param height    the heigth in rows of the component
 */
private void addPanel( Component component,
                       int row, int column, int width, int height ){
  constraints.gridx = column;
  constraints.gridy = row;
  constraints.gridwidth = width;
```

```
  constraints.gridheight = height;
  // set constraints and add component
  layout.setConstraints( component, constraints );
  container.add( component );
} //end addPanel() method

/**
 * Adds a panel to a container
 *
 * @param container the container to add the panel
 */
public void addPanel( Container container ){
  JPanel dummy = new JPanel();
  dummy.setBackground( BACKGROUND_COLOR );
  container.add( dummy );
} //end addPanel() method

/**
 * Opens the streams for communicating with the client
 *
 * @throws IOException when the outpu/input stream can not be openned
 */
private void openStreams() throws IOException{

  // open the sreams needed
  inObjectFromServer = new ObjectInputStream( clientSocket.getInputStream() );
  outObjectToServer = new ObjectOutputStream( clientSocket.getOutputStream() );

} // end openStreams() method

/**
 * Selects one from the valid server to try connecting
 *
 * @return a string IP address of the server
 */
private String selectServer(){

  String serverToReturn;
  double random = 10 * Math.random();
  int randomIndex = ( int ) random;
  serverToReturn = VALID_SERVERS[randomIndex % VALID_SERVERS.length];
  return serverToReturn;

} // end selectServer() method

/**
```

```
 * Selects one from the valid server to try connecting
 *
 * @return a string IP address of the server
 * @throws IOException if the input/output streams fail
    * @throws ClassNotFoundException if the received object is not of a known class
 */
private ArrayList getServerAvailableFiles() throws IOException,
   ClassNotFoundException{

 boolean exit = false;
 boolean connected = false;
 int serverIndex = 0;

 if( serverMode == "Auto select" ){
  while( !connected && ( serverIndex < VALID_SERVERS.length ) ){
   serverSelected = VALID_SERVERS[serverIndex];
   System.out.println( "server : " + serverSelected );
   serverIndex++;
   try{
    clientSocket = new Socket( serverSelected, FTP_PORT );
    connected = true;
   } catch( IOException ioe ){
   } //end try catch block
  } //end while
  if( !connected ){
   clientSocket = new Socket( serverSelected, FTP_PORT );
  }
 } else{
  clientSocket = new Socket( serverMode, FTP_PORT );
 }

 openStreams();
 connectionStatus = CONNECTED;
 transferStatusPanel.updateFileTransferStatus();
 fileRequest = new PFTPFileRequest( FILE_ARRAY_REQUEST_TYPE,
                                    new String(),
                                    new String(),
                                    0,
                                    0 );

 //send the file request object to the server
 outObjectToServer.writeObject( fileRequest );
 outObjectToServer.flush();
 Vector filesAtServer = ( Vector ) inObjectFromServer.readObject();
 serverAvailableFiles = new ArrayList();
 for( int i = 0; i < filesAtServer.size(); i++ ){
```

```
    serverAvailableFiles.add( filesAtServer.elementAt( i ) );
   }
   System.out.println( "Files at server : " + serverAvailableFiles );
   clientSocket.close();

   connectionStatus = DISCONNECTED;
   transferStatusPanel.updateFileTransferStatus();

   return serverAvailableFiles;

  } // end selectServer() method

  /**
   * Sends a file request package to the server asking for a file
   *
   * @param fileRequest the file request
   * @throws Exception
   */
  public void askForFile( PFTPFileRequest fileRequest ) throws Exception{

   // some variable initialization
   boolean fileNotFound = false;
   fileTransfered = false;
   cancelFileTransfer = false;
   packetCounter = 0;
   int byteCounter = 0;
   int numOfBytes = 1;
   int percent = 0;
   numOfPackets = 1;
   fileNameToAsk = fileRequest.getFile();
   boolean firstIteration = true;

   // create a file to receive
   File file = new File( fileNameToAsk );

   int connectionCounter = 1;
   int imagePacketUpdate = 0;
   int imagePacketUpdateCounter = 1;

   //update the file status panel
   transferStatusPanel.updateFileTransferStatus();

   if( retrieveFailedTransfers ){
    offset = fileRequest.getOffset();
    fileHash = fileRequest.getHash();
    packetCounter = offset;
```

```
     numOfPackets = packetCounter + 1;
    } //end if

    // The loop that keep asking the file till gets it all
    while( !cancelFileTransfer && !fileTransfered && !fileNotFound &&
          ( packetCounter < numOfPackets ) &&
          (       connectionCounter       <=       (       MAX_RECONNECTION_TRIES       *
MAX_TOTAL_TRIES ) ) ){

     if(  (  (  MAX_RECONNECTION_TRIES  *  MAX_TOTAL_TRIES  )  %
connectionCounter ) ==
          1 ){
       delay( 2000 );
     } else{
       delay( 2000 );
     } //end if else
     connectionCounter++;

     try{
       if( serverMode.equals( new String( "Auto select" ) ) ){
         //select a valid server to try connect
         if( !firstIteration ){

           serverSelected = selectServer();

         }
       } else{

         serverSelected = serverMode;

       } //end if

       System.out.println( "Trying to connect to the server...." +
                           serverSelected );
       delay( 2000 );

       //connect to the server selected and set the time out period
       clientSocket = new Socket( serverSelected, FTP_PORT );
       clientSocket.setSoTimeout( SO_TIME_OUT );

       connectionStatus = CONNECTED;
       transferStatusPanel.updateFileTransferStatus();

       System.out.println( "Client connected!" );

       try{
```

```
  openStreams();
} catch( Exception ioe ){

  JOptionPane.showMessageDialog( null, "Client: Failed to open streams",
                                 "Exception",
                                 JOptionPane.ERROR_MESSAGE, null );
} // end try catch block

fileRequest.setOffset( offset );
fileRequest.setHash( fileHash );

//send the file request object to the server
outObjectToServer.writeObject( fileRequest );
outObjectToServer.flush();

//receive the file hash value if it is asked for the first time
fileHash = inObjectFromServer.readUTF();
System.out.println( "Hash value received " + fileHash );

if( !fileHash.equals( new String( "fileNotFound" ) ) &&
    !fileHash.equals( new String( "hashProblem" ) ) &&
    !fileHash.equals( new String( "fileNotMatch" ) ) ){

  //receive the file size from the server
  fileSize = inObjectFromServer.readInt();
  System.out.println( "File size to receive " + fileSize + " bytes" );
  numOfPackets = ( fileSize / packetSize ) + 1;
  imagePacketUpdate = numOfPackets / ( IMAGE_UPDATE_TIMES - 1 );

  //update the transfer status panel
  transferStatusPanel.updateFileTransferStatus();

  if( packetCounter == 0 ){

    //initialize a byte array for the file bytes
    fileInBytes = new byte[fileSize + packetSize];

  } else if( retrieveFailedTransfers && firstIteration ){
    firstIteration = false;
    byte[] partialBytes = ( byte[] ) oldPartialTransfersBytes.get(
       fileNameToAsk );
    ByteArrayOutputStream baos =
       new ByteArrayOutputStream( fileSize + packetSize );
    baos.write( partialBytes, 0, partialBytes.length );
    fileInBytes = baos.toByteArray();
    baos.close();
```

```
  byteCounter = packetSize * offset;
} //end if else if

byte[] newPacket = new byte[packetSize];

try{
  if( packetCounter == 0 ){
    startTime = System.currentTimeMillis();
  } //end if

  fileIsTransfering = true;

  //sets the maximum of the transfer progress bar
  transferProgressPanel.setProgressBarMax( numOfPackets );

  try{
    //receiving the packets and updating the file byte array
    while( !cancelFileTransfer && ( packetCounter < numOfPackets ) ){ //end if
      offset = packetCounter;
      inObjectFromServer.read( newPacket );

      transferProgressPanel.updateProgressBar( packetCounter );

      for( int i = 0; i < packetSize; i++ ){
        fileInBytes[byteCounter + i] = newPacket[i];
      } //end for loop
      System.out.println( "Bytes " + ( byteCounter + 1 ) +
                          " to " + ( byteCounter + packetSize ) +
                          " received" );
      if( ( packetCounter % imagePacketUpdate ) == 0 ){
        //update image progress panel
        imageProgressPanel.updateImageProgress( fileInBytes );
      } //and if
      totalFileTransferTime = System.currentTimeMillis() - startTime;
      delay( 500 );
      byteCounter += packetSize;
      packetCounter++;
    } //end while loop
  } catch( NullPointerException npe ){
    System.out.println( "Problem with null pointers" );
  }

  //renews the maximum of the transfer progress bar
  transferProgressPanel.setProgressBarMax( numOfPackets - 1 );

} catch( IOException ioe ){ //in case of connection loss
```

```
  System.out.println( "Problem reading I/O Streams" );
  connectionStatus = DISCONNECTED;
  connectionLossTimes++;
  transferStatusPanel.updateFileTransferStatus();
 } //end try catch block

 stopTime = System.currentTimeMillis();

 //create the file and pass the byte array formed in it
 if( !cancelFileTransfer && packetCounter == numOfPackets ){
  fileTransfered = true;
  outObjectToServer.writeBoolean( true );
  outObjectToServer.flush();
  if( !file.createNewFile() ){
   file.delete();
  } //end if
  file = new File( fileNameToAsk );
  file.createNewFile();
  FileOutputStream editFile = new FileOutputStream( file );
  editFile.write( fileInBytes );

  totalFileTransferTime = stopTime - startTime;

  editFile.close();
  if( retrieveFailedTransfers ){
   deletePartialFileRequest( fileNameToAsk );
   retrieveFailedTransfers = false;
   deletePartialFileRequest( fileNameToAsk );
  } //end if
  transferProgressPanel.updateProgressBar( packetCounter - 1 );
  transferStatusPanel.updateFileTransferStatus();
  imageProgressPanel.updateImageProgress( fileInBytes );
  imageProgressPanel.updateImageProgress( fileInBytes );

  JOptionPane.showMessageDialog( null,
                        "File successfully transfered in\n" +
                        "            " +
                        totalFileTransferTime + " msec", null,
                        JOptionPane.INFORMATION_MESSAGE, null );
  resetApplication();

 } else{
  outObjectToServer.writeBoolean( false );
  outObjectToServer.flush();

  System.out.println( "File: " + fileNameToAsk +
```

```
                        " transfer interrupted!" );
          } //end if else
        } else if( fileHash.equals( new String( "fileNotMatch" ) ) ){
          System.out.println( "Server doesn't have the file version asked!" );
        } //end if else

        try{
          System.out.println( "Close socket" );
          clientSocket.close();
        } catch( IOException ioe ){

        } //end try catch block
      } catch( Exception ce ){
        connectionStatus = TRYING_TO_CONNECT;
        transferStatusPanel.updateFileTransferStatus();
        if( connectionCounter == ( MAX_RECONNECTION_TRIES *
MAX_TOTAL_TRIES + 1 ) ){
          JOptionPane.showMessageDialog( null,
                                "The server couldn't be reached ", null,
                                JOptionPane.ERROR_MESSAGE, null );
        } //end if
      } //end try catch block

      if( connectionStatus == TRYING_TO_CONNECT ){
        System.out.println( "Connection failed!" );
      } //end if
      firstIteration = false;
    } //end while loop

    if( !fileTransfered && !cancelFileTransfer ){
      fileRequest.setOffset( offset );
      System.out.println( "fileHash saved" + fileHash );
      fileRequest.setHash( fileHash );
      savePartialFileRequest( fileRequest,
                          fileInBytes );

      JOptionPane.showMessageDialog( null, " File transfer failed!\n " +
          "The partial file data stored for future use", null,
          JOptionPane.INFORMATION_MESSAGE, null );
      resetApplication();
    } //end if

  } //end askForFile() method

  /**
   * Cancels the file's transfer
```

```
 */
public void cancelFileTransfer(){
  cancelFileTransfer = true;
} //end setFileNameToAsk() method

/**
 * Sets the file name to ask the server
 *
 * @param fileName the string file name
 */
public void setFileNameToAsk( String fileName ){
  fileNameToAsk = fileName;
} //end setFileNameToAsk() method

/**
 * Starts the file dowmnloading
 *
 * @param ask true if the ask the file button is pressed
 */
public void setAskTheFile( boolean ask ){
  askTheFile = ask;
} //end setFileNameToAsk() method

/**
 * Sets the server mode of the file transfer
 *
 * @param mode the string server mode
 */
public void setServerMode( String mode ){
  serverMode = mode;
} //end setServerMode() method

/**
 * Sets the packet size for the file transfer
 *
 * @param sizeOfThePacket the packet size selected for the file transfer
 */
public void setPacketSize( int sizeOfThePacket ){
  packetSize = sizeOfThePacket;
} //end setPacketSize() method

/**
 * Sets the offset for the file transfer
 *
 * @param theOffset the offset value
 */
```

```
public void setOffset( int theOffset ){
  offset = theOffset;
} //end setPacketSize() method

/**
 * Returns the name of the file asked from the server
 *
 * @return the name of the file asked from the server
 */
public synchronized String getServerSelected(){
  notify();
  return serverSelected;
} //end getFileNameToAsk() method

/**
 * Returns the name of the file asked from the server
 *
 * @return the name of the file asked from the server
 */
public synchronized String getFileNameToAsk(){
  notify();
  return fileNameToAsk;
} //end getFileNameToAsk() method

/**
 * Returns the size of the file asked from the server
 *
 * @return the size of the file asked from the server
 */
public synchronized int getPacketSize(){
  notify();
  return packetSize;
} //end getPacketSize() method

/**
 * Returns the size in bytes of the file asked
 *
 * @return the size ij bytes of the file asked
 */
public synchronized int getFileSize(){
  notify();
  return fileSize;
} //end getFileSize() method

/**
 * Returns the number of packet needed for the transfer
```

```
 *
 * @return the number of packets need for the transfer
 */
public synchronized int getNumOfPackets(){
  notify();
  return numOfPackets;
} //end getNumOfPackets() method

/**
 * Returns the number of packet already received
 *
 * @return the number of packet already received
 */
public synchronized int getReceivedPackets(){
  notify();
  return packetCounter;
} //end getReceivedPackets() method

/**
 * Returns the file bytes as byte array
 *
 * @return the file bytes as byte array
 */
public synchronized byte[] getFileInBytes(){
  notify();
  return fileInBytes;
} //end getFileInBytes() method

/**
 * Returns the connection status
 *
 * @return the connection status
 */
public synchronized String getConnectionStatus(){
  notify();
  return connectionStatus;
} //end getConnectionStatus() method

/**
 * Returns the connection loss times
 *
 * @return the connection loss times
 */
public synchronized int getConnectionLossTimes(){
  notify();
  return connectionLossTimes;
```

```
} //end getConnectionLossTimes() method

/**
 * Returns the total file transfer time
 *
 * @return the total file transfer time
 */
public synchronized long getTotalFileTransferTime(){
  notify();
  return totalFileTransferTime;
} //end getTotalFileTransferTime() method

/**
 * Returns the total file transfer time
 *
 * @return the total file transfer time
 */
public synchronized boolean getFileTransfered(){
  notify();
  return fileTransfered;
} //end getTotalFileTransferTime() method

/**
 * Returns the total file transfer time
 *
 * @return the total file transfer time
 */
public synchronized ControlPanel getControlPanel(){
  notify();
  return controlPanel;
} //end getTotalFileTransferTime() method

/**
 * Return a titled JtextPanel with specific attributes
 *
 * @param titleText the delay time in milliseconds
 * @param charSize the character size
 * @param charColor the character color
 * @param backgroundColor the backgroundColor
 * @param isUnderlined if the character is underlined or not
 * @param isBold if the character is bold or not
 * @return a titled JtextPanel with specific attributes
 */
public JTextPane getTitledPanel( String titleText, int charSize,
                        Color charColor, Color backgroundColor,
                        boolean isUnderlined, boolean isBold ){
```

```
  JTextPane textPane;
  DefaultStyledDocument doc = new DefaultStyledDocument();
  textPane = new JTextPane( doc );
  MutableAttributeSet attr = new SimpleAttributeSet();
  //StyleConstants.setBackground(attr, Color.blue);
  StyleConstants.setFontSize( attr, charSize );
  StyleConstants.setUnderline( attr, isUnderlined );
  StyleConstants.setForeground( attr, charColor );
  StyleConstants.setBold( attr, isBold );
  textPane.setAlignmentX( JTextPane.CENTER_ALIGNMENT );
  textPane.setAlignmentY( JTextPane.CENTER_ALIGNMENT );
  textPane.setEditable( false );
  textPane.setEnabled( false );
  textPane.setBackground( BACKGROUND_COLOR );
  textPane.setCharacterAttributes( attr, true );
  textPane.setText( titleText );
  return textPane;
} //end getTitledPanel() method

/**
 * Returns the boolean value of file transfering status
 *
 * @return true if file is still transferring, false otherwise
 */
public boolean getFileIsTransfering(){
  return fileIsTransfering;
}

/**
 * Saves a partial file request to file request database for future use
 *
 * @param partialFileRequest the partial file request to save in  database
 * @param partialFileBytes the partial file bytes to save in database
 */
public void savePartialFileRequest( PFTPFileRequest partialFileRequest,
                                    byte[] partialFileBytes ){

  oldPartialTransferRequests.put( partialFileRequest.getFile(),
                                  partialFileRequest );
  oldPartialTransfersBytes.put( partialFileRequest.getFile(),
                                partialFileBytes );
  oldPartialTransfersFiles.add( partialFileRequest.getFile() );

} //end savePartialFileRequest() method

/**
```

```
 * Deletes a partial file request from file request database
 *
 * @param partialFileNameRequested the name of the file of the
 *                                 request to delete from database
 */
public void deletePartialFileRequest( String partialFileNameRequested ){

  oldPartialTransferRequests.remove( partialFileNameRequested );
  oldPartialTransfersBytes.remove( partialFileNameRequested );
  oldPartialTransfersFiles.remove( partialFileNameRequested );

} //end deletePartialFileRequest() method

/**
 * Resets the client application
 */
public void resetApplication(){

  fileSize = 0;
  fileHash = "";
  connectionStatus = DISCONNECTED;
  connectionLossTimes = 0;
  totalFileTransferTime = 0;

  controlPanel.reset();

  transferProgressPanel.reset();

  transferStatusPanel.reset();

  imageProgressPanel.reset();

} //end resetApplication() method

/**
 * Stops the panel threads of the downloading panels
 */
public void startCheckToAsk(){
  while( true ){
    delay( 50 );
    if( askTheFile ){
      try{
        askForFile( fileRequest );
      } catch( Exception e ){

      }
```

```
        askTheFile = false;
      }
    }
  } //end

  /**
   * Delay the execution of the code
   *
   * @param msec the delay time in milliseconds
   */
  public void delay( int msec ){
    try{
      Thread.sleep( msec );
    } catch( InterruptedException ie ){

    }
  } //end delay() method

  /**
   * The main method
   *
   * @param args the arguments
   * @throws Exception if exception occurs in FTPClient application
   */
  public static void main( String args[] ) throws Exception{

    PFTPClient client = new PFTPClient();
    client.startCheckToAsk();

  } //end main method

//----------------------------------------
//
//    Private Inner Classes:
//
//----------------------------------------

  /**
   * Provide a control panel to control the application
   */
  private class ControlPanel extends JPanel{

    /** The PFTPClient class object */
    private PFTPClient controlPanelClient;

    /** The panel that shows the transfer progress */
```

```
private TransferProgressPanel progressPanel;

/** The panel that shows the transfer status during downloading */
private TransferStatusPanel statusPanel;

/** The panel that displays the image during the downloading */
private ImageProgressPanel imagePanel;

/** The title for the control panel  */
private JPanel controlPanelTitle;

/** The FileSelectionDialog object for file choice  */
private FileSelectionDialog fileChoiceDialog;

/** The text field for file choice  */
private JTextField fileChoiceField;

/** The combo box for server mode choice  */
private JComboBox serverModeChoiceBox;

/** The combo box for packet size choice  */
private JComboBox packetSizeChoiceBox;

/** The button group for file choice  */
private ButtonGroup fileButtonGroup;

/** The button group for server choice  */
private ButtonGroup serverButtonGroup;

/**
 * The radio button that give the option to retrieve
 * previous partial transferred files
 */
private JRadioButton previousFailedFiles;

/**
 * The radio button that give the option to retrieve
 * files from a PFTP server
 */
private JRadioButton filesFromServer;

/**
 * The radio button that give the option to the user
 * to enter the server IP manually
 */
private JRadioButton manualServerSelection;
```

```
/**
 * The radio button that give the option to the user
 * to from the predefined PFTP servers
 */
private JRadioButton predefinedServerSelection;

/** The button to a server manual  */
private JButton selectServer;

/** The button to retreive the availiable file at PFTP Server  */
private JButton retreiveAvailableFiles;

/** The button to ask for the file transfer  */
private JButton askTheFile;

/** The button to cancel the file transfer  */
private JButton cancelTransfer;

/** The text field to enter manual the server to ask the file from  */
private JTextField manualSelectServer;

/** The server IP array with servers can be chosen  */
private String[] serverIPs;

/** The packet size array with the packet sizes can be chosen  */
private String[] packetSizes;

/** The file selected  */
private String fileSelection;

/** The server selected  */
private String serverSelection;

/** The packet size selected  */
private String packetSizeSelection;

/** The boolean value if a file selected or not  */
private boolean fileSelected = false;

/** The boolean value if a server selected or not  */
private boolean serverSelected = false;

/** The boolean value if a packet size selected or not  */
private boolean packetSizeSelected = false;
```

```
  /**
   * The default constructor
   *
   * @param panelClient the FTPClientVer7 object assosiated with
   * @param progressPanel the TransferProgressPanel object assosiated with
   * @param statusPanel the TransferStatusPanel object assosiated with
   * @param imagePanel the ImageProgressPanel object assosiated with
   */
  public ControlPanel( PFTPClient panelClient,
                       TransferProgressPanel progressPanel,
                       TransferStatusPanel statusPanel,
                       ImageProgressPanel imagePanel ){

    super();

    serverIPs = new String[VALID_SERVERS.length + 2];
    for( int i = 0; i < serverIPs.length; i++ ){
      if( i == 0 ){
        serverIPs[i] = "Select server";
      } else if( i == 1 ){
        serverIPs[i] = "Auto select";
      } else{
        serverIPs[i] = VALID_SERVERS[i - 2];
      } //end if else if
    } //end for loop

    packetSizes = new String[PACKET_SIZES.length + 1];
    for( int i = 0; i < packetSizes.length; i++ ){
      if( i == 0 ){
        packetSizes[i] = "Select packet size";
      } else{
        packetSizes[i] = PACKET_SIZES[i - 1];
      } //end if else if
    } //end for loop

    controlPanelClient = panelClient;
    progressPanel = progressPanel;
    statusPanel = statusPanel;
    imagePanel = imagePanel;

    setBorder( BorderFactory.createLineBorder( PANEL_BORDER_COLOR ) );
    setBackground( BACKGROUND_COLOR );
    setLayout( new GridLayout( 8, 3 ) );

    add( getTitledPanel( "CONTROL PANEL", PANEL_TITLE_FONT_SIZE,
FONT_COLOR,
```

```
                BACKGROUND_COLOR, true, true ) );

// just to fill the empty cells
controlPanelClient.addPanel( this );
controlPanelClient.addPanel( this );

// construct textfield with default sizing
add( getTitledPanel( "File to retreive:", PANEL_LABEL_FONT_SIZE,
                FONT_COLOR,
                BACKGROUND_COLOR, false, true ) );

fileButtonGroup = new ButtonGroup();
previousFailedFiles = new JRadioButton( "Previous failures", false );
filesFromServer = new JRadioButton( "PFTP Server files", true );

//inner class to handle JRadioButton events
ItemListener fileMode = new ItemListener(){

  // handle JRadioButton event
  public void itemStateChanged( ItemEvent event ){
    // determine whether check box selected
    if( event.getSource() == previousFailedFiles ){
      String[] fileList = new String[oldPartialTransfersFiles.size() + 1];
      for( int i = 0; i < fileList.length; i++ ){
        if( i == 0 ){
          fileList[i] = "Select file";
        } else{
          fileList[i] = ( String ) oldPartialTransfersFiles.get( i - 1 );
        } //end if else
      } //end for loop
      if( previousFailedFiles.isSelected() &&
          !oldPartialTransfersFiles.isEmpty() ){
        retrieveFailedTransfers = true;
        fileChoiceDialog = new FileSelectionDialog( controlPanelClient,
          controlPanelClient.getControlPanel(),
          fileChoiceField,
          fileList );
        packetSizeSelected = true;
      } else if( previousFailedFiles.isSelected() &&
              oldPartialTransfersFiles.isEmpty() ){
        JOptionPane.showMessageDialog( null,
          "No file transfer failure by now!\n" +
          "Select a file from the PFTP servers.",
          null,
          JOptionPane.INFORMATION_MESSAGE,
          null );
```

```
        manualServerSelection.setEnabled( false );
        predefinedServerSelection.setEnabled( false );
        manualSelectServer.setEnabled( false );
        selectServer.setEnabled( false );
        serverModeChoiceBox.setEnabled( false );
        retreiveAvailableFiles.setEnabled( false );
        packetSizeChoiceBox.setEnabled( false );
        askTheFile.setEnabled( false );
      }
    } else if( event.getSource() == filesFromServer ){
      manualServerSelection.setEnabled( true );
      predefinedServerSelection.setEnabled( true );
      serverModeChoiceBox.setEnabled( true );
      askTheFile.setEnabled( true );
    } //end if else
  } //end itemstateChanged

}; // end anonymous inner class

previousFailedFiles.addItemListener( fileMode );
previousFailedFiles.setBackground( BACKGROUND_COLOR );
previousFailedFiles.setForeground( FONT_COLOR );

filesFromServer.addItemListener( fileMode );
filesFromServer.setBackground( BACKGROUND_COLOR );
filesFromServer.setForeground( FONT_COLOR );

fileButtonGroup.add( previousFailedFiles );
fileButtonGroup.add( filesFromServer );

add( previousFailedFiles );
add( filesFromServer );

// construct textfield with default sizing
add( getTitledPanel( "Server to receive from:", PANEL_LABEL_FONT_SIZE,
                FONT_COLOR,
                BACKGROUND_COLOR, false, true ) );

serverButtonGroup = new ButtonGroup();
manualServerSelection = new JRadioButton( "Manual Selection", false );
predefinedServerSelection = new JRadioButton( "Existing Servers", true );

//inner class to handle JRadioButton events
ItemListener serverListener = new ItemListener(){

  // handle JRadioButton event
```

```
  public void itemStateChanged( ItemEvent event ){
    // determine whether a radio button selected
    if( event.getSource() == manualServerSelection ){
      manualSelectServer.setEnabled( true );
      serverModeChoiceBox.setEnabled( false );
      selectServer.setEnabled( true );
    } else if( event.getSource() == predefinedServerSelection ){
      manualSelectServer.setEnabled( false );
      selectServer.setEnabled( false );
      serverModeChoiceBox.setEnabled( true );
    } //end if else
  } //end itemstateChanged

}; // end anonymous inner class

manualServerSelection.addItemListener( serverListener );
manualServerSelection.setBackground( BACKGROUND_COLOR );
manualServerSelection.setForeground( FONT_COLOR );

predefinedServerSelection.addItemListener( serverListener );
predefinedServerSelection.setBackground( BACKGROUND_COLOR );
predefinedServerSelection.setForeground( FONT_COLOR );

serverButtonGroup.add( manualServerSelection );
serverButtonGroup.add( predefinedServerSelection );

add( manualServerSelection );
add( predefinedServerSelection );

// construct textfield with default sizing
add( getTitledPanel( "Add Server IP:", PANEL_LABEL_FONT_SIZE,
                FONT_COLOR,
                BACKGROUND_COLOR, false, true ) );

manualSelectServer = new JTextField( "Enter server IP" );
manualSelectServer.setEditable( true );
manualSelectServer.setEnabled( false );
add( manualSelectServer );

// constructs the select server button
selectServer = new JButton( "Select the server" );
selectServer.setFont( new Font( null, Font.BOLD, PANEL_LABEL_FONT_SIZE ) );
selectServer.setForeground( FONT_COLOR );
selectServer.setHorizontalTextPosition( JButton.LEFT );
selectServer.setBackground( BACKGROUND_COLOR );
selectServer.setEnabled( false );
```

```
selectServer.addActionListener( new ActionListener(){
  public void actionPerformed( ActionEvent event ){

    try{
      String serverIP = manualSelectServer.getText();
      InetAddress.getByName( serverIP );
      if( !retrieveFailedTransfers ){
        retreiveAvailableFiles.setEnabled( true );
      }
      packetSizeChoiceBox.setEnabled( false );
      controlPanelClient.setServerMode( serverIP );
      serverSelected = true;
    } catch( UnknownHostException uhe ){
      JOptionPane.showMessageDialog( null,
                                     "Select a valid server IP address!",
                                     null,
                                     JOptionPane.INFORMATION_MESSAGE,
                                     null );
    }

  } //end actionPerformed() method
} //end ActionListener class
); //end addActionListener() method

add( selectServer );

// construct textfield with default sizing
add( getTitledPanel( "Choose server mode:", PANEL_LABEL_FONT_SIZE,
                     FONT_COLOR,
                     BACKGROUND_COLOR, false, true ) );

// construct textfield with default sizing
serverModeChoiceBox = new JComboBox( serverIPs );
serverModeChoiceBox.setFont( new Font( null, Font.BOLD,
                                       PANEL_LABEL_FONT_SIZE ) );
serverModeChoiceBox.setBackground( BACKGROUND_COLOR );
serverModeChoiceBox.setForeground( FONT_COLOR );
serverModeChoiceBox.setEnabled( true );
serverModeChoiceBox.addItemListener(

    // anonymous inner class to handle JComboBox events
    new ItemListener(){

  // handle JComboBox event
  public void itemStateChanged( ItemEvent event ){
    // determine whether check box selected
```

```
    if( event.getStateChange() == ItemEvent.SELECTED ){
      serverSelection = serverIPs[serverModeChoiceBox.getSelectedIndex()];
      serverSelected = true;
      if( !retrieveFailedTransfers ){
        retreiveAvailableFiles.setEnabled( true );
      }
    }
    controlPanelClient.setServerMode( serverSelection );
  }
} // end anonymous inner class

); // end call to addItemListener
add( serverModeChoiceBox );

// constructs the retreive files button
retreiveAvailableFiles = new JButton( "Retreive files.. " );
retreiveAvailableFiles.setFont( new Font( null, Font.BOLD,
                                PANEL_LABEL_FONT_SIZE ) );
retreiveAvailableFiles.setForeground( FONT_COLOR );
retreiveAvailableFiles.setHorizontalTextPosition( JButton.LEFT );
retreiveAvailableFiles.setBackground( BACKGROUND_COLOR );
retreiveAvailableFiles.addActionListener( new ActionListener(){
  public void actionPerformed( ActionEvent event ){
    try{
      serverAvailableFiles = controlPanelClient.getServerAvailableFiles();
      retreiveAvailableFiles.setEnabled( false );
      if( !serverAvailableFiles.isEmpty() ){
        String[] fileList = new String[serverAvailableFiles.size() + 1];
        for( int i = 0; i < fileList.length; i++ ){
          if( i == 0 ){
            fileList[i] = "Select file";
          } else{
            fileList[i] = ( String ) serverAvailableFiles.get( i - 1 );
          } //end if else
        } //end for loop
        fileChoiceDialog = new FileSelectionDialog( controlPanelClient,
            controlPanelClient.getControlPanel(),
            fileChoiceField,
            fileList );
        packetSizeChoiceBox.setEnabled( true );
      } else{
        JOptionPane.showMessageDialog( null,
                                "No files availiable at server",
                                null,
                                JOptionPane.INFORMATION_MESSAGE,
                                null );
```

```
            } //end if else
           } catch( IOException ioe ){
            JOptionPane.showMessageDialog( null,
               "Fail to retreive file names from PFTP server",
               "Exception",
               JOptionPane.ERROR_MESSAGE, null );
            retreiveAvailableFiles.setEnabled( false );

           } catch( ClassNotFoundException cnfe ){
            JOptionPane.showMessageDialog( null, "Unknown object received",
                                  "Exception",
                                  JOptionPane.ERROR_MESSAGE, null );
            retreiveAvailableFiles.setEnabled( false );
           } //end try catch block

         } //end actionPerformed() method
        } //end ActionListener class
        ); //end addActionListener() method

        retreiveAvailableFiles.setEnabled( false );
        add( retreiveAvailableFiles );

        // construct textfield with default sizing
        add( getTitledPanel( "File selected :", PANEL_LABEL_FONT_SIZE,
                        FONT_COLOR,
                        BACKGROUND_COLOR, false, true ) );

        // construct textfield with default sizing
        fileChoiceField = new JTextField( "No file selected" );
        fileChoiceField.setFont( new Font( null, Font.BOLD, PANEL_LABEL_FONT_SIZE
    ) );
        fileChoiceField.setBackground( BACKGROUND_COLOR );
        fileChoiceField.setForeground( FONT_COLOR );
        fileChoiceField.setEditable( false );

        add( fileChoiceField );

        // construct packet size choice box
        packetSizeChoiceBox = new JComboBox( packetSizes );
        packetSizeChoiceBox.setFont( new Font( null, Font.BOLD,
                                      PANEL_LABEL_FONT_SIZE ) );
        packetSizeChoiceBox.setBackground( BACKGROUND_COLOR );
        packetSizeChoiceBox.setForeground( FONT_COLOR );
        packetSizeChoiceBox.setEnabled( true );
        packetSizeChoiceBox.addItemListener(
```

```
    // anonymous inner class to handle JComboBox events
    new ItemListener(){

   // handle JComboBox event
   public void itemStateChanged( ItemEvent event ){
     // determine whether check box selected
     if( event.getStateChange() == ItemEvent.SELECTED ){
       packetSizeSelection = packetSizes[packetSizeChoiceBox.
          getSelectedIndex()];
       System.out.println( "packet size selected.. " + packetSizeSelection );
       controlPanelClient.setPacketSize( new Integer( packetSizeSelection ).
                                  intValue() );
       packetSizeSelected = true;
       askTheFile.setEnabled( true );
     }

   }

} // end anonymous inner class

); // end call to addItemListener

packetSizeChoiceBox.setEnabled( false );
add( packetSizeChoiceBox );

controlPanelClient.addPanel( this );

// constructs the download button
askTheFile = new JButton( "Download   ",
                    new ImageIcon( "downloadArrow.gif" ) );
askTheFile.setFont( new Font( null, Font.BOLD, PANEL_LABEL_FONT_SIZE ) );
askTheFile.setForeground( FONT_COLOR );
askTheFile.setHorizontalTextPosition( JButton.LEFT );
askTheFile.setBackground( BACKGROUND_COLOR );
askTheFile.addActionListener( new ActionListener(){
  public void actionPerformed( ActionEvent event ){
    if( fileSelected && serverSelected && packetSizeSelected ){
      // change enabled status of panel elements
      filesFromServer.setEnabled( false );
      previousFailedFiles.setEnabled( false );
      manualServerSelection.setEnabled( false );
      predefinedServerSelection.setEnabled( false );
      manualSelectServer.setEnabled( false );
      selectServer.setEnabled( false );
      serverModeChoiceBox.setEnabled( false );
      retreiveAvailableFiles.setEnabled( false );
```

```
  packetSizeChoiceBox.setEnabled( false );
  askTheFile.setEnabled( false );
  cancelTransfer.setEnabled( true );

  // create a file request object
  controlPanelClient.setOffset( 0 );
  fileSelection = fileChoiceField.getText();
  if( !retrieveFailedTransfers ){
    fileRequest = new PFTPFileRequest( FILE_REQUEST_TYPE,
                                fileSelection,
                                new String( "" ), 0,
                                new Integer(
        packetSizeSelection ).
                                intValue() );
  } else{
    fileRequest = ( PFTPFileRequest )
        oldPartialTransferRequests.get( fileSelection );

  } //end if else

  try{
    controlPanelClient.setAskTheFile( true );
  } catch( Exception e ){
    JOptionPane.showMessageDialog( null, "Problem in asking the file" +
                              fileNameToAsk, "Exception",
                              JOptionPane.ERROR_MESSAGE );
  } //end try catch block

} else if( !fileSelected ){
  if( serverSelected && packetSizeSelected ){
    JOptionPane.showMessageDialog( null,
        "Please select server, file and packet size first!" );
  } else if( !serverSelected && !packetSizeSelected ){
    JOptionPane.showMessageDialog( null,
        "Please select server, file and packet size first!" );
  } else if( !serverSelected ){
    JOptionPane.showMessageDialog( null,
        "Please select the server and a file first!" );
  } else if( !packetSizeSelected ){
    JOptionPane.showMessageDialog( null,
        "Please select a file and a packet size first!" );
  } //end if else if
} else if( serverSelected && !packetSizeSelected ){
  JOptionPane.showMessageDialog( null,
                          "Please select packet size first!" );
} else if( !serverSelected && packetSizeSelected ){
```

```
          JOptionPane.showMessageDialog( null,
                                "Please select server first!" );
        } else if( !serverSelected && !packetSizeSelected ){
          JOptionPane.showMessageDialog( null,
             "Please select the server and packet size first!" );
        } //end if else if
       } //end actionPerformed() method
      } //end ActionListener class
      ); //end addActionListener() method
      askTheFile.setEnabled( false );

      add( askTheFile );

      // constructs the stop transfer button
      cancelTransfer = new JButton( "Cancel Download" );
      cancelTransfer.setFont( new Font( null, Font.BOLD, PANEL_LABEL_FONT_SIZE
) );
      cancelTransfer.setForeground( FONT_COLOR );
      cancelTransfer.setHorizontalTextPosition( JButton.LEFT );
      cancelTransfer.setBackground( BACKGROUND_COLOR );
      cancelTransfer.setEnabled( false );
      cancelTransfer.addActionListener( new ActionListener(){
       public void actionPerformed( ActionEvent event ){
        int option = JOptionPane.showOptionDialog( null,
          "Are you sure you want to cancel " +
          " download?", "Warning", JOptionPane.YES_NO_OPTION,
          JOptionPane.QUESTION_MESSAGE, null, null
          , null );

        if( option == JOptionPane.OK_OPTION ){
          controlPanelClient.cancelFileTransfer();
          resetApplication();

        } //end if

       } //end actionPerformed() method
      } //end ActionListener class
      ); //end addActionListener() method

      add( cancelTransfer );

      controlPanelClient.addPanel( this );
      controlPanelClient.addPanel( this );

      setVisible( true );
```

```
  } //end constructor

  /**
   * Sets the fileSelected value
   *
   * @param state the the fileSelected boolean value
   */
  public void setFileSelected( boolean state ){

    fileSelected = state;

  } //end setFileSelected() method

  /**
   * Resets the panel
   */
  public void reset(){
    //reset all the panel components
    filesFromServer.setEnabled( true );
    filesFromServer.setSelected( true );
    previousFailedFiles.setEnabled( true );
    manualServerSelection.setEnabled( true );
    predefinedServerSelection.setEnabled( true );
    predefinedServerSelection.setSelected( true );
    manualSelectServer.setEnabled( false );
    selectServer.setEnabled( false );
    serverModeChoiceBox.setEnabled( true );
    retreiveAvailableFiles.setEnabled( false );
    packetSizeChoiceBox.setEnabled( false );
    askTheFile.setEnabled( true );
    cancelTransfer.setEnabled( false );
    fileSelected = false;
    serverSelected = false;
    packetSizeSelected = false;
    fileIsTransfering = false;
  } //end reset() method
} //end  ControlPanel class

/**
 * Provide a transfer progress panel to monitor the progress of file transfer
 */
private class TransferProgressPanel extends JPanel{

  /** The PFTP Client of the transfer progress panel */
  private PFTPClient transferProgressPanelClient;
```

```
/** The download message panel  */
private JPanel downloadMessagePanel;

/** The download message  */
private JLabel downloadMessage;

/** The progress bar panel  */
private JPanel progressBarPanel;

/** The progress bar  */
private JProgressBar progressBar;

/** The panel with the download percentage panel  */
private JPanel downloadPercentagePanel;

/** The file downloading message  */
private JLabel downloadPercentage;

/** The file downloading message  */
private String fileToTransfer;

/**
 * The default constructor
 *
 * @param client the PFTPClient object
 */
public TransferProgressPanel( PFTPClient client ){
  super();

  transferProgressPanelClient = client;

  setBorder( BorderFactory.createLineBorder( PANEL_BORDER_COLOR ) );
  setBackground( BACKGROUND_COLOR );
  setLayout( new GridLayout( 5, 1 ) );

  // construct textfield with default sizing
  add( getTitledPanel( "TRANSFER PROGRESS", PANEL_TITLE_FONT_SIZE,
                  FONT_COLOR,
                  BACKGROUND_COLOR, true, true ) );

  transferProgressPanelClient.addPanel( this );

  downloadMessagePanel = new JPanel();
  downloadMessagePanel.setBackground( BACKGROUND_COLOR );
  downloadMessage = new JLabel( "No file is downloading" );
```

```
    downloadMessage.setFont( new Font( null, Font.BOLD,
PANEL_LABEL_FONT_SIZE ) );
    downloadMessage.setForeground( FONT_COLOR );
    downloadMessagePanel.add( downloadMessage );
    add( downloadMessagePanel );

    progressBar = new JProgressBar();
    progressBar.setBackground( new Color( 207, 233, 245 ) );
    //progressBar.setBorderPainted(true);
    add( progressBar );

    downloadPercentagePanel = new JPanel();
    downloadPercentagePanel.setBackground( BACKGROUND_COLOR );
    downloadPercentage = new JLabel();
    downloadPercentage.setFont( new Font( null, Font.BOLD,
                                PANEL_LABEL_FONT_SIZE ) );
    downloadPercentage.setForeground( FONT_COLOR );
    downloadPercentagePanel.add( downloadPercentage );
    add( downloadPercentagePanel );

    setVisible( true );
   } // end constructor

   /**
    * Sets the progress bar
    *
    * @param max the progress bar max
    */
   public void setProgressBarMax( int max ){
    progressBar.setMinimum( 0 );
    progressBar.setMaximum( max );
   } //end setProgressBarMax() method

   /**
    * Updates the progress bar
    *
    * @param value the progress bar value
    */
   public void updateProgressBar( int value ){

    if( transferProgressPanelClient.getFileIsTransfering() ){
      SwingUtilities.invokeLater( new UpdateLabel( downloadMessage,
         "Downloading file " +
         transferProgressPanelClient.getFileNameToAsk() ) );

    } else{
```

```
      SwingUtilities.invokeLater( new UpdateLabel( downloadMessage,
         "No file is downloding" ) );

    }
    SwingUtilities.invokeLater( new UpdateProgressBar( progressBar, value ) );
    SwingUtilities.invokeLater( new UpdateLabel( downloadPercentage,
       Math.floor( ( ( double ) value /
                progressBar.getMaximum() ) * 100 ) + " %" ) );

  } //end updateProgressBar() method

  /**
   * Resets the transfer progress panel
   */
  public void reset(){
    SwingUtilities.invokeLater( new UpdateLabel( downloadMessage,
       "No file is downloding" ) );
    SwingUtilities.invokeLater( new UpdateProgressBar( progressBar, 0 ) );
    SwingUtilities.invokeLater( new UpdateLabel( downloadPercentage, "" ) );

  } //end reset() method

} //end TransferProgressPanel class

/**
 * Provide a transfer status panel to monitor the file transfer status
 */
private class TransferStatusPanel extends JPanel{

  /** The panel that shows the transfer status */
  private PFTPClient transferStatusPanelClient;

  /** The transfer status panel title  */
  private JTextPane transferStatusPanelTitle;

  /** The file asked info  */
  private JTextPane fileAskedInfo;

  /** The file size info  */
  private JTextPane fileSizeInfo;

  /** The packet size used info  */
  private JTextPane packetSizeUsedInfo;

  /** The connection status info  */
  private JTextPane connectionStatusInfo;
```

```
/** The connection loss number info  */
private JTextPane connectionLossNumberInfo;

/** The total file transfer time info  */
private JTextPane totalFileTransferTimeInfo;

/**
 * The default constructor
 *
 * @param client the PFTPClient object
 */
public TransferStatusPanel( PFTPClient client ){
  super();

  transferStatusPanelClient = client;

  setBorder( BorderFactory.createLineBorder( PANEL_BORDER_COLOR ) );

  fileAskedInfo = getTitledPanel( "No file asked", PANEL_LABEL_FONT_SIZE,
                            FONT_COLOR,
                            BACKGROUND_COLOR, false, true );
  fileAskedInfo.setForeground( FONT_COLOR );

  fileSizeInfo = getTitledPanel( "", PANEL_LABEL_FONT_SIZE,
                           FONT_COLOR,
                           BACKGROUND_COLOR, false, true );
  fileSizeInfo.setForeground( FONT_COLOR );

  packetSizeUsedInfo = getTitledPanel( "", PANEL_LABEL_FONT_SIZE,
                               FONT_COLOR,
                               BACKGROUND_COLOR, false, true );
  packetSizeUsedInfo.setForeground( FONT_COLOR );

  connectionStatusInfo = getTitledPanel( "Disconnected",
                                PANEL_LABEL_FONT_SIZE, FONT_COLOR,
                                BACKGROUND_COLOR, false, true );
  connectionStatusInfo.setForeground( FONT_COLOR );

  connectionLossNumberInfo = getTitledPanel( "0",
                                   PANEL_LABEL_FONT_SIZE,
                                   FONT_COLOR,
                                   BACKGROUND_COLOR, false, true );
  connectionLossNumberInfo.setForeground( FONT_COLOR );

  totalFileTransferTimeInfo = getTitledPanel( "0 msec",
```

```
                                PANEL_LABEL_FONT_SIZE,
                                FONT_COLOR,
                                BACKGROUND_COLOR, false, true );
    totalFileTransferTimeInfo.setForeground( FONT_COLOR );

    setBackground( BACKGROUND_COLOR );
    setLayout( new GridLayout( 7, 2 ) );

    add( getTitledPanel( "TRANSFER STATUS", PANEL_TITLE_FONT_SIZE,
FONT_COLOR,
                BACKGROUND_COLOR, true, true ) );

    add( getTitledPanel( "", PANEL_LABEL_FONT_SIZE, FONT_COLOR,
                BACKGROUND_COLOR, false, true ) );

    add( getTitledPanel( "File asked :", PANEL_LABEL_FONT_SIZE, FONT_COLOR,
                BACKGROUND_COLOR, false, true ) );

    add( fileAskedInfo );

    add( getTitledPanel( "File size :", PANEL_LABEL_FONT_SIZE, FONT_COLOR,
                BACKGROUND_COLOR, false, true ) );

    add( fileSizeInfo );

    add( getTitledPanel( "Packet size used :", PANEL_LABEL_FONT_SIZE,
                FONT_COLOR,
                BACKGROUND_COLOR, false, true ) );

    add( packetSizeUsedInfo );

    add( getTitledPanel( "Connection status :", PANEL_LABEL_FONT_SIZE,
                FONT_COLOR,
                BACKGROUND_COLOR, false, true ) );

    add( connectionStatusInfo );

    add( getTitledPanel( "Connection loss number :", PANEL_LABEL_FONT_SIZE,
                FONT_COLOR,
                BACKGROUND_COLOR, false, true ) );

    add( connectionLossNumberInfo );

    add( getTitledPanel( "Total transfer time:", PANEL_LABEL_FONT_SIZE,
                FONT_COLOR,
                BACKGROUND_COLOR, false, true ) );
```

```
  add( totalFileTransferTimeInfo );

  setVisible( true );
  this.setEnabled( false );

} //end constructor

/**
 * Updates file transfer's status
 */
public void updateFileTransferStatus(){

  SwingUtilities.invokeLater( new UpdateTextPane( fileAskedInfo,
     transferStatusPanelClient.getFileNameToAsk() ) );

  SwingUtilities.invokeLater( new UpdateTextPane( fileSizeInfo,
     new Integer(
     transferStatusPanelClient.getFileSize() ).toString() ) );

  SwingUtilities.invokeLater( new UpdateTextPane( packetSizeUsedInfo,
     new Integer(
     transferStatusPanelClient.getPacketSize() ).toString() ) );

  String connectStatus = transferStatusPanelClient.getConnectionStatus();
  if( connectStatus.equals( CONNECTED ) ){
    SwingUtilities.invokeLater( new UpdateTextPane( connectionStatusInfo,
       connectStatus + " with " +
       transferStatusPanelClient.getServerSelected() ) );
  } else{
    SwingUtilities.invokeLater( new UpdateTextPane( connectionStatusInfo,
       connectStatus ) );
  }

  SwingUtilities.invokeLater( new UpdateTextPane( connectionLossNumberInfo,
     new Integer(
     transferStatusPanelClient.getConnectionLossTimes() ).toString() ) );

  SwingUtilities.invokeLater( new UpdateTextPane( totalFileTransferTimeInfo,
     new Long(
     transferStatusPanelClient.getTotalFileTransferTime() ).toString() +
     " msec" ) );

} //end updateFileTransferStatus() method

/**
```

```
   * Resets the panel
   */
  public void reset(){

    SwingUtilities.invokeLater( new UpdateTextPane( fileAskedInfo,
       "No file asked" ) );

    SwingUtilities.invokeLater( new UpdateTextPane( fileSizeInfo, "" ) );

    SwingUtilities.invokeLater( new UpdateTextPane( packetSizeUsedInfo, "" ) );

    SwingUtilities.invokeLater( new UpdateTextPane( connectionStatusInfo,
       "Disconnected" ) );

    SwingUtilities.invokeLater( new UpdateTextPane( connectionLossNumberInfo,
       "0" ) );

    SwingUtilities.invokeLater( new UpdateTextPane( totalFileTransferTimeInfo,
       "0 sec" ) );

  } //end reset() method

}

/**
 * Provide a image progress panel to view the partial file transferred
 */
private class ImageProgressPanel extends JPanel{

  /** The PFTPClient object assosiated with this panel */
  private PFTPClient imageProgressPanelClient;

  /** The image progress label  */
  private JLabel imageProgressLabel;

  /** The updated times  */
  private int updatedTimes;

  /** The fileNameToPreview  */
  private String fileNameToPreview;

  /**
   * Constructor
   *
   * @param client the PFTPClient object assosiated with this panel
   */
```

```
public ImageProgressPanel( PFTPClient client ){
  super();

  imageProgressPanelClient = client;

  updatedTimes = 0;

  fileNameToPreview = null;

  setBorder( BorderFactory.createLineBorder( PANEL_BORDER_COLOR ) );
  setBackground( BACKGROUND_COLOR );
  setLayout( new BorderLayout( 2, 1 ) );

  Icon noPreview = new ImageIcon( LOGO_IMAGE );
  imageProgressLabel = new JLabel( noPreview );
  imageProgressLabel.setForeground( FONT_COLOR );
  imageProgressLabel.setHorizontalTextPosition( JLabel.CENTER );
  imageProgressLabel.setVerticalTextPosition( JLabel.BOTTOM );
  add( imageProgressLabel );

  setVisible( true );
} //end constructor

/**
 * Updates the iamage progress panel
 *
 * @param fileBytes the file byte array to display
 */
public void updateImageProgress( byte[] fileBytes ){

  try{

    byte[] currentImageBytes = fileBytes;
    int fileLength = currentImageBytes.length;
    byte[] endOfFile = ( new String( ";" ) ).getBytes();
    int endOfFileLength = endOfFile.length;
    ByteArrayOutputStream baos = new ByteArrayOutputStream();
    baos.write( currentImageBytes, 0, fileLength );
    baos.write( endOfFile, 0, endOfFileLength );
    currentImageBytes = baos.toByteArray();
    baos.close();

    SwingUtilities.invokeLater( new UpdateImageLabel( imageProgressLabel,
       "Preview of file " + imageProgressPanelClient.getFileNameToAsk(),
       new ImageIcon( currentImageBytes ) ) );
```

```
    } catch( Exception e ){
      System.out.println( e );
    }

  } ///end updateImageProgress() method

  /**
   * Resets the panel
   */
  public void reset(){
    SwingUtilities.invokeLater( new UpdateImageLabel( imageProgressLabel,
        "",
        new ImageIcon( LOGO_IMAGE ) ) );

  } //end reset() method
} //end ImageProgressPanel

/**
 * Updates a label given dynamically
 */
public class UpdateLabel extends Thread{

  /** The label to update  */
  private JLabel label;

  /** The text to set on the label  */
  private String text;

  /**
   * Constructor
   * @param l the label
   * @param t the text to set on the label
   */
  public UpdateLabel( JLabel l, String t ){
    label = l;
    text = t;
  } //end constructor

  // method called to update label
  public void run(){
    label.setText( text );
  } //end run() method

} // end class UpdateLabel

/**
```

```
 * Updates a image label given dynamically
 */
public class UpdateImageLabel extends Thread{

  /** The label to uadate */
  private JLabel label;

  /** The text to set on the label  */
  private String text;

  /** The icon to set on the label   */
  private Icon icon;

  /**
   * Constructor
   * @param l the label
   * @param t the text to set on the label
   * @param i the icon to set on the label
   */
  public UpdateImageLabel( JLabel l, String t, ImageIcon i ){
    label = l;
    text = t;
    icon = i;
  } //end constructor

  // method called to update label
  public void run(){
    label.setText( text );
    label.setIcon( icon );
  } //end run() method

} // end class UpdateImageLabel

/**
 * Updates a image text pane given dynamically
 */
public class UpdateTextPane extends Thread{

  /** The text pane to update */
  private JTextPane textPane;

  /** The text to set on the text pane  */
  private String text;

  /**
   * Constructor
```

```
   * @param jtp the text tp update
   * @param t the text to set on the text pane
   */
  public UpdateTextPane( JTextPane jtp, String t ){
    textPane = jtp;
    text = t;
  } //end constructor

  // method called to update text pane
  public void run(){
    textPane.setText( text );
  } //end run() method

} // end class UpdateTextPane

/**
 * Updates a progess bar
 */
public class UpdateProgressBar extends Thread{

  /** The progress bar to update  */
  private JProgressBar progressBar;

  /** The value to set for the progress bar  */
  private int progressBarValue;

  /**
   * Constructor
   * @param bar the progress bar to update
   * @param value the value to set for te progress bar
   */
  public UpdateProgressBar( JProgressBar bar, int value ){
    progressBar = bar;
    progressBarValue = value;
  } //end constructor

  // method called to update progress bar
  public void run(){
    progressBar.setValue( progressBarValue );
  } //end run() method

} // end class UpdateProgressBar

/**
 * Displays a dialog with a file list to choose from
 */
```

```
private class FileSelectionDialog extends JDialog{

  /** The dialog client */
  private PFTPClient dialogClient;

  /** The dialog control panel */
  private ControlPanel dialogControlPanel;

  /** The dialog text field */
  private JTextField dialogTextField;

  /** The item list  */
  private String[] itemList;

  /** The dialog combo box  */
  private JComboBox dialogComboBox;

  /** The dialog button  */
  private JButton dialogButton;

  /** The dialog selection  */
  private String dialogSelection;

  /**
   * Constructor
   *
   * @param client the PFTPClientPPC object assosiated with
   * @param panel the ControlPanel object assosiated with
   * @param fileField the dialog field to set
   * @param list the list of items to choose
   */
  public FileSelectionDialog( PFTPClient client,
                              ControlPanel panel,
                              JTextField fileField,
                              String[] list
                              ){
    super( client, "Choose a file to transfer", true );

    dialogClient = client;
    dialogControlPanel = panel;
    dialogTextField = fileField;
    itemList = list;
    dialogSelection = null;
    setBackground( BACKGROUND_COLOR );
    setSize( 300, 180 );
    this.setLocation( 392, 134 );
```

```
      Container c = getContentPane();
      c.setLayout( new GridLayout( 5, 3 ) );
      // just to fill the empty cell

      for( int i = 0; i < 4; i++ ){
        dialogClient.addPanel( c );
      }

      // construct textfield with default sizing
      dialogComboBox = new JComboBox( itemList );
      dialogComboBox.setFont( new Font( null, Font.BOLD,
                                  PANEL_LABEL_FONT_SIZE ) );
      dialogComboBox.setBackground( BACKGROUND_COLOR );
      dialogComboBox.setForeground( FONT_COLOR );
      dialogComboBox.setEnabled( true );
      dialogComboBox.addItemListener(

         // anonymous inner class to handle JComboBox events
         new ItemListener(){

        // handle JComboBox event
        public void itemStateChanged( ItemEvent event ){
          // determine whether check box selected
          if( event.getStateChange() == ItemEvent.SELECTED ){
            dialogSelection = itemList[dialogComboBox.
                getSelectedIndex()];
          }

        }

      } // end anonymous inner class

      ); // end call to addItemListener

      c.add( dialogComboBox );

      for( int i = 0; i < 5; i++ ){
        dialogClient.addPanel( c );
      }

      // constructs the ok button
      dialogButton = new JButton( "OK" );
      dialogButton.setFont( new Font( null, Font.BOLD, PANEL_LABEL_FONT_SIZE )
);
      dialogButton.setForeground( FONT_COLOR );
      dialogButton.setHorizontalTextPosition( JButton.LEFT );
```

```
    dialogButton.setBackground( BACKGROUND_COLOR );
    dialogButton.addActionListener( new ActionListener(){
      public void actionPerformed( ActionEvent event ){
        if( dialogSelection == null ){
          dialogControlPanel.setFileSelected( false );
          JOptionPane.showMessageDialog( null,
                                         "No file selected!",
                                         null,
                                         JOptionPane.INFORMATION_MESSAGE,
                                         null );

        } else{
          dialogTextField.setText( dialogSelection );
          dialogControlPanel.setFileSelected( true );
          setVisible( false );
        } //end if else
      } //end actionPerformed() method
    } //end ActionListener class
    ); //end addActionListener() method

    c.add( dialogButton );

    for( int i = 0; i < 4; i++ ){
      dialogClient.addPanel( c );
    }

    show();
   } //end constructor

  } //end FileSelectionDialog class

} //end PFTPClient class
```

```
//packages for network componets
import java.io.*;
import java.net.*;

//packages for user interface components
import java.awt.*;
import java.awt.event.*;
import java.util.*;

//package for security components
import java.security.*;

//package for charecter encoding components
import sun.misc.*;

/**
 * File PFTPClientPPC.java
 *
 * This is a FTP Client for the Pocket PC version
 *
 * This client also ask for a file and stays in a loop keep asking
 * until it gets it all. In the first request it receives the
 * hash value of the requested file so when the connection is
 * lost client can ask an other server the same file with out
 * worrying about the authenticity of it.
 *
 * @author LT Periklis Pantoleon
 * @version 1.0
 */
class PFTPClientPPC extends Frame{

//----------------------------------------
//
//    Private Data Members:
//
//----------------------------------------

  /** The default port where the FTP client is asking the server */
  private static final int FTP_PORT = 6789;

  /** The string array with the valid servers can be chosen  */
  private static final String[] VALID_SERVERS = {
     "127.0.0.1", "131.120.105.237", "131.120.105.250", "131.120.201.46"};

  /** The packet size array with the packet sizes can be chosen  */
  private String[] PACKET_SIZES = {new String( "32" ),
```

```
    new String( "64" ), new String( "128" ),
    new String( "256" ), new String( "512" ),
    new String( "1024" )};

/** The connected status */
private static final String CONNECTED = "Connected";

/** The disconnected status */
private static final String DISCONNECTED = "Disconnected";

/** The status when the client is trying to connect */
private static final String TRYING_TO_CONNECT = "Trying to connect.....";

/** The timeout period of the TCP sockets  */
private static final int SO_TIME_OUT = 3000;

/** The length in bytes of the file hash value */
private static final int MD5_HASH_LENGTH = 16;

/** The font size for the panels' titles */
private static final int PANEL_TITLE_FONT_SIZE = 17;

/** The font size for the panels' labels */
private static final int PANEL_LABEL_FONT_SIZE = 12;

/** The panels' font color */
private static final Color FONT_COLOR = Color.white;

/** The color of the backgroung of the frames and panels */
private static final Color BACKGROUND_COLOR = new Color( 77, 92, 240 );

/** The color of the panels' border */
private static final Color PANEL_BORDER_COLOR = Color.blue;

/** The maximum reconnection tries each time client lose connection*/
private static final int MAX_RECONNECTION_TRIES = 3;

/** The maximum reconnection tries each time client lose connection */
private static final int MAX_TOTAL_TRIES = 3;

/** The times tht the image is updated in the image progress panel */
private static final int IMAGE_UPDATE_TIMES = 10;

/** The request type when a file is asked */
private static final int FILE_REQUEST_TYPE = 1;
```

```
/** The request type when a file array is asked */
private static final int FILE_ARRAY_REQUEST_TYPE = 0;

/** The type for the message dialog */
private static final int MESSAGE_DIALOG_TYPE = 1;

/** The type for the yes-no dialog */
private static final int YES_NO_DIALOG_TYPE = 0;

/** The message dialog OK option */
private static final int YES_OPTION = 1;

/** The message dialog NO option */
private static final int NO_OPTION = 0;

/** The file name of the logo image of the client frame */
private static final String LOGO_IMAGE = "pftpLogoClient.gif";

//-------------------------------------------------
//
//    Data Members for the FTP Client functionality:
//
//-------------------------------------------------

/** The socket client use to communicate with the server  */
private Socket clientSocket;

/** The object input stream to receive objects from client */
private ObjectInputStream inObjectFromServer;

/** The object output stream to send objects to client */
private ObjectOutputStream outObjectToServer;

/** The file request received from client */
private PFTPFileRequest fileRequest;

/** The vector that store that old partial file bytes */
private Vector oldPartialTransferRequests;

/** The vector that store that old partial file bytes */
private Vector oldPartialTransfersBytes;

/** The vector of the file names that failed to downloaded */
private Vector oldPartialTransfersFiles;

/** The vector list of the files available on PFTP server */
```

```
private Vector serverAvailableFiles;

/** The name of the asked file */
private String fileNameToAsk;

/** The server mode selected */
private String serverMode;

/** The server selected */
private String serverSelected;

/** The hash value of the file asked */
private String fileHash;

/** The size in bytes of the file asked */
private int fileSize;

/** The offset from where we need the file */
private int offset;

/** The packet size used for the file transfer */
private int packetSize;

/** The file as a byte array */
private byte[] fileInBytes = null;

/** The packet counter */
private int packetCounter;

/** The number of packets used for the file*/
private int numOfPackets;

/** The start time of the file reception */
private long startTime;

/** The end time of the file reception */
private long stopTime;

/** The total file transfer time */
private long totalFileTransferTime;

/** The connection status */
private String connectionStatus;

/** The connection loss times */
private int connectionLossTimes;
```

```
/** The connection loss times */
private int messageOption;

/** The boolean value of the intetion of the user to ask a file */
private boolean wantFile;

/** The boolean value if the file is transfered or not */
private boolean fileTransfered;

/** The boolean value the file transfer is canceled or not */
private boolean cancelFileTransfer;

/** The boolean value if user wants to retreive failed transferred files  */
private boolean retrieveFailedTransfers;

/** The file name for the partial retreived file */
private String partialRetreivedFileName;

//-------------------------------------------------
//
//    Data Members for the User Interface:
//
//-------------------------------------------------

/** The container of the main frame */
private Container container;

/** The the layout manager of the container */
private GridLayout layout;

/** The control panel */
private ControlPanel controlPanel;

/** The panel that shows the transfer status during downloading */
//private TransferStatusPanel transferStatusPanel;

/** The panel that displays the image during the downloading */
private ImageProgressPanel imageProgressPanel;

/** The boolean tranfering status */
private boolean fileIsTransfering = false;

/** The boolean value if user want to download the file or not */
private boolean askTheFile = false;
```

```
//----------------------------------------
//
//    Constructor:
//
//----------------------------------------

  /**
   * Default constructor
   */
  public PFTPClientPPC(){

    super( "PFTP PocketPC 1.0" );

    FrontDialog fd = new FrontDialog( this );

    // set the layout of the container
    layout = new GridLayout( 1, 1 );
    setLayout( layout );

    // add the panels on the main frame's panels

    add( controlPanel );

    addWindowListener( new WindowListener(){
      public void windowOpened( WindowEvent event ){
      }

      public void windowClosed( WindowEvent event ){
        System.exit( 0 );
      }

      public void windowClosing( WindowEvent event ){
        System.exit( 0 );
      }

      public void windowIconified( WindowEvent event ){
      }

      public void windowDeiconified( WindowEvent event ){
      }

      public void windowActivated( WindowEvent event ){
      }

      public void windowDeactivated( WindowEvent event ){
      }
```

```
  } );

  //sets other attributes of the main frame
  setSize( 240, 300 );
  setVisible( true );

  //initialization of the private members
  oldPartialTransferRequests = new Vector();
  oldPartialTransfersBytes = new Vector();
  oldPartialTransfersFiles = new Vector();
  serverAvailableFiles = new Vector();

  fileNameToAsk = null;
  fileSize = 0;
  packetSize = 0;
  fileHash = "";
  connectionStatus = DISCONNECTED;
  retrieveFailedTransfers = false;
  connectionLossTimes = 0;
  totalFileTransferTime = 0;
  askTheFile = false;

} // end default constructor

/**
 * Opens the streams for communicating with the client
 *
 * @throws IOException when the outpu/input stream can not be openned
 */
private void openStreams() throws IOException{

  // open the sreams needed
  inObjectFromServer = new ObjectInputStream( clientSocket.getInputStream() );
  outObjectToServer = new ObjectOutputStream( clientSocket.getOutputStream() );

} // end openStreams() method

/**
 * Selects one from the valid server to try connecting
 *
 * @return a string IP address of the server
 */
private String selectServer(){

  String serverToReturn;
```

```
  double random = 10 * Math.random();
  int randomIndex = ( int ) random;
  serverToReturn = VALID_SERVERS[randomIndex % VALID_SERVERS.length];
  return serverToReturn;

} // end selectServer() method

/**
 * Selects one from the valid server to try connecting
 *
 * @return a string IP address of the server
 * @throws IOException
 * @throws ClassNotFoundException
 */
private Vector getServerAvailableFiles() throws IOException,
    ClassNotFoundException{

  boolean exit = false;
  boolean connected = false;
  int serverIndex = 0;

  if( serverMode == "Auto select" ){
    while( !connected && ( serverIndex < VALID_SERVERS.length ) ){
      serverSelected = VALID_SERVERS[serverIndex];
      System.out.println( "server : " + serverSelected );
      serverIndex++;
      try{
        clientSocket = new Socket( serverSelected, FTP_PORT );
        connected = true;
      } catch( IOException ioe ){
      } //end try catch block
    } //end while
    if( !connected ){
      clientSocket = new Socket( serverSelected, FTP_PORT );
    }
  } else{
    clientSocket = new Socket( serverMode, FTP_PORT );
  }

  openStreams();
  connectionStatus = CONNECTED;
  fileRequest = new PFTPFileRequest( FILE_ARRAY_REQUEST_TYPE,
                                     new String(),
                                     new String(),
                                     0,
                                     0 );
```

```
  //send the file request object to the server
  outObjectToServer.writeObject( fileRequest );
  outObjectToServer.flush();
  Vector filesAtServer = ( Vector ) inObjectFromServer.readObject();
  serverAvailableFiles = new Vector();
  for( int i = 0; i < filesAtServer.size(); i++ ){
    serverAvailableFiles.add( filesAtServer.elementAt( i ) );
  }
  System.out.println( "Files at server : " + serverAvailableFiles );
  clientSocket.close();

  connectionStatus = DISCONNECTED;

  return serverAvailableFiles;

} // end selectServer() method

/**
 * Sends a file request package to the server asking for a file
 *
 * @param fileRequest the file request
 * @throws Exception
 */
public void askForFile( PFTPFileRequest fileRequest ) throws Exception{
  // some variable initialization
  boolean fileNotFound = false;
  fileTransfered = false;
  cancelFileTransfer = false;
  packetCounter = 0;
  int byteCounter = 0;
  int numOfBytes = 1;
  int percent = 0;
  numOfPackets = 1;
  fileNameToAsk = fileRequest.getFile();
  boolean firstIteration = true;

  // create a file to receive
  fileNameToAsk = fileRequest.getFile();
  File file = new File( fileNameToAsk );

  int connectionCounter = 1;
  int imagePacketUpdate = 0;
  int imagePacketUpdateCounter = 1;

  if( retrieveFailedTransfers ){
```

```
    offset = fileRequest.getOffset();
    fileHash = fileRequest.getHash();
    packetCounter = offset;
    numOfPackets = packetCounter + 1;
   } //end if

   // The loop that keep asking the file till gets it all
   while( !cancelFileTransfer && !fileTransfered && !fileNotFound &&
        ( packetCounter < numOfPackets ) &&
        ( connectionCounter <= ( MAX_RECONNECTION_TRIES *
MAX_TOTAL_TRIES ) ) ){

    System.out.println( "while" );
    if( ( ( MAX_RECONNECTION_TRIES * MAX_TOTAL_TRIES ) %
connectionCounter ) ==
       1 ){
     delay( 2000 );
    } else{
     delay( 2000 );
    } //end if else
    connectionCounter++;

    try{

     if( serverMode.equals( new String( "Auto select" ) ) ){
      //select a valid server to try connect
      if( !firstIteration ){

       serverSelected = selectServer();

      }
     } else{

      serverSelected = serverMode;

     } //end if

     System.out.println( "Trying to connect to the server...." + serverMode );
     delay( 1000 );

     //connect to the server selected and set the time out period
     clientSocket = new Socket( serverSelected, FTP_PORT );
     clientSocket.setSoTimeout( SO_TIME_OUT );

     connectionStatus = CONNECTED;
```

```
System.out.println( "Client connected!" );

try{
  openStreams();
} catch( Exception ioe ){

  MessageDialog md = new MessageDialog( this,
                                controlPanel,
                                "** Exception **",
                                MESSAGE_DIALOG_TYPE,
     "Client: Failed to open streams" );

} // end try catch block

fileRequest.setOffset( offset );
fileRequest.setHash( fileHash );

//send the file request object to the server
outObjectToServer.writeObject( fileRequest );
outObjectToServer.flush();

imageProgressPanel = new ImageProgressPanel( this );

if( offset == 0 ){
  //receive the file hash value if it is asked for the first time
  fileHash = inObjectFromServer.readUTF();
  System.out.println( "Hash value received " + fileHash );
} //end if

if( !fileHash.equals( new String( "fileNotFound" ) ) &&
    !fileHash.equals( new String( "hashProblem" ) ) &&
    !fileHash.equals( new String( "fileNotMatch" ) ) ){

  //receive the file size from the server
  fileSize = inObjectFromServer.readInt();
  System.out.println( "File size to receive " + fileSize + " bytes" );
  numOfPackets = ( fileSize / packetSize ) + 1;
  imagePacketUpdate = numOfPackets / ( IMAGE_UPDATE_TIMES - 1 );

  if( packetCounter == 0 ){

    //initialize a byte array for the file bytes
    fileInBytes = new byte[fileSize + packetSize];

  } else if( retrieveFailedTransfers && firstIteration ){
    firstIteration = false;
```

```
  byte[] partialBytes = ( byte[] ) oldPartialTransfersBytes.elementAt(
     oldPartialTransfersFiles.indexOf( fileNameToAsk ) );
  ByteArrayOutputStream baos =
     new ByteArrayOutputStream( fileSize + packetSize );
  baos.write( partialBytes, 0, partialBytes.length );
  fileInBytes = baos.toByteArray();
  baos.close();
  byteCounter = packetSize * offset;
} //end if else if

byte[] newPacket = new byte[packetSize];

try{
  if( packetCounter == 0 ){
   startTime = System.currentTimeMillis();
  } //end if

  fileIsTransfering = true;

  //receiving the packets and updating the file byte array
  while( !cancelFileTransfer && ( packetCounter < numOfPackets ) ){ //end if
    offset = packetCounter;
    inObjectFromServer.read( newPacket );

    for( int i = 0; i < packetSize; i++ ){
      fileInBytes[byteCounter + i] = newPacket[i];
    } //end for loop
    System.out.println( "Bytes " + ( byteCounter + 1 ) +
                        " to " + ( byteCounter + packetSize ) +
                        " received" );

    percent = ( int ) Math.floor( ( ( double ) byteCounter /
                                  fileSize ) * 100 );
    imageProgressPanel.setTitle( percent + "% of " + fileNameToAsk +
                                 " received!" );
    if( ( packetCounter % imagePacketUpdate ) == 0 ){
      //update image progress panel
      imageProgressPanel.updateImageProgress( fileInBytes );
    } //and if

    delay( 500 );
    byteCounter += packetSize;
    percent = ( int ) Math.floor( ( ( double ) byteCounter /
                                  fileSize ) * 100 );
    if( percent > 100 )
      percent = 100;
```

```
    imageProgressPanel.setTitle( percent + "% of " + fileNameToAsk +
                                 " received!" );
    packetCounter++;
   } //end while loop

 } catch( IOException ioe ){ //in case of connection loss
   System.out.println( "Problem reading I/O Streams" );
   connectionStatus = DISCONNECTED;
   connectionLossTimes++;
 } //end try catch block

 stopTime = System.currentTimeMillis();

 //create the file and pass the byte array formed in it
 if( !cancelFileTransfer && packetCounter == numOfPackets ){
   fileTransfered = true;
   outObjectToServer.writeBoolean( true );
   outObjectToServer.flush();

   if( file.exists() ){
     file.delete();
   } else{

     imageProgressPanel.updateImageProgress( fileInBytes );
     imageProgressPanel.updateImageProgress( fileInBytes );

     file = new File( fileNameToAsk );
     FileOutputStream editFile = new FileOutputStream( file );
     editFile.write( fileInBytes );
     totalFileTransferTime = stopTime - startTime;
     editFile.close();
     if( retrieveFailedTransfers ){
       deletePartialFileRequest( fileNameToAsk );
       retrieveFailedTransfers = false;
       deletePartialFileRequest( fileNameToAsk );
     } //end if

     MessageDialog md = new MessageDialog( this,
         controlPanel,
         "** Message **",
         MESSAGE_DIALOG_TYPE,
         "File transfered in " +
         totalFileTransferTime + " msec!" );
     if( oldPartialTransfersFiles.contains( fileNameToAsk ) ){
       int index = oldPartialTransfersFiles.indexOf( fileNameToAsk );
       oldPartialTransferRequests.removeElementAt( index );
```

```
            oldPartialTransfersBytes.removeElementAt( index );
            oldPartialTransfersFiles.removeElementAt( index );
           }

           imageProgressPanel.setVisible( false );

           resetApplication();
          }

         } else{
          System.out.println( "File: " + fileNameToAsk +
                        " transfer interrupted!" );
         } //end if else
        } else{
         System.out.println( "Server doesn't have the file version asked!" );
        } //end if else

        try{
         System.out.println( "Close socket" );
         clientSocket.close();
        } catch( IOException ioe ){

        } //end try catch block
       } catch( Exception ce ){
        connectionStatus = TRYING_TO_CONNECT;
        if(      connectionCounter      ==      (      MAX_RECONNECTION_TRIES      *
MAX_TOTAL_TRIES + 1 ) ){
         MessageDialog md = new MessageDialog( this,
                                    controlPanel,
                                    "** ERROR **",
                                    MESSAGE_DIALOG_TYPE,
           "The server couldn't be reached " );

        } //end if
       } //end try catch block
     } //end while loop

     if( !fileTransfered && !cancelFileTransfer ){
      fileRequest.setOffset( offset );
      fileRequest.setHash( fileHash );
      savePartialFileRequest( fileRequest,
                        fileInBytes );

      MessageDialog md = new MessageDialog( this,
                                    controlPanel,
                                    "** ERROR **",
```

```
                                        MESSAGE_DIALOG_TYPE,
        "Transfer failed.Partial data stored!" );

  } //end if

} //end askForFile() method

/**
 * Cancels the file's transfer
 */
public void cancelFileTransfer(){
  cancelFileTransfer = true;
} //end setFileNameToAsk() method

/**
 * Sets the file name to ask the server
 *
 * @param fileName the string file name
 */
public void setFileNameToAsk( String fileName ){
  fileNameToAsk = fileName;
} //end setFileNameToAsk() method

/**
 * Starts the file dowmnloading
 *
 * @param ask true if the ask the file button is pressed
 */
public void setAskTheFile( boolean ask ){
  askTheFile = ask;
} //end setFileNameToAsk() method

/**
 * Sets the server mode of the file transfer
 *
 * @param mode the string server mode
 */
public void setServerMode( String mode ){
  serverMode = mode;
} //end setServerMode() method

/**
 * Sets the packet size for the file transfer
 *
 * @param sizeOfThePacket the packet size selected for the file transfer
 */
```

```
public void setPacketSize( int sizeOfThePacket ){
  packetSize = sizeOfThePacket;
} //end setPacketSize() method

/**
 * Sets the offset for the file transfer
 *
 * @param theOffset the offset value
 */
public void setOffset( int theOffset ){
  offset = theOffset;
} //end setPacketSize() method

/**
 * Returns the name of the file asked from the server
 *
 * @return the name of the file asked from the server
 */
public synchronized String getServerMode(){
  notify();
  return serverMode;
} //end getFileNameToAsk() method

/**
 * Returns the name of the file asked from the server
 *
 * @return the name of the file asked from the server
 */
public synchronized String getFileNameToAsk(){
  notify();
  return fileNameToAsk;
} //end getFileNameToAsk() method

/**
 * Returns the size of the file asked from the server
 *
 * @return the size of the file asked from the server
 */
public synchronized int getPacketSize(){
  notify();
  return packetSize;
} //end getPacketSize() method

/**
 * Returns the size in bytes of the file asked
 *
```

```
 * @return the size in bytes of the file asked
 */
public synchronized int getFileSize(){
  notify();
  return fileSize;
} //end getFileSize() method

/**
 * Returns the number of packet needed for the transfer
 *
 * @return the number of packets need for the transfer
 */
public synchronized int getNumOfPackets(){
  notify();
  return numOfPackets;
} //end getNumOfPackets() method

/**
 * Returns the number of packet already received
 *
 * @return the number of packet already received
 */
public synchronized int getReceivedPackets(){
  notify();
  return packetCounter;
} //end getReceivedPackets() method

/**
 * Returns the file bytes as byte array
 *
 * @return the file bytes as byte array
 */
public synchronized byte[] getFileInBytes(){
  notify();
  return fileInBytes;
} //end getFileInBytes() method

/**
 * Returns the connection status
 *
 * @return the connection status
 */
public synchronized String getConnectionStatus(){
  notify();
  return connectionStatus;
} //end getConnectionStatus() method
```

```
/**
 * Returns the connection loss times
 *
 * @return the connection loss times
 */
public synchronized int getConnectionLossTimes(){
  notify();
  return connectionLossTimes;
} //end getConnectionLossTimes() method

/**
 * Returns the total file transfer time
 *
 * @return the total file transfer time
 */
public synchronized long getTotalFileTransferTime(){
  notify();
  return totalFileTransferTime;
} //end getTotalFileTransferTime() method

/**
 * Returns the total file transfer time
 *
 * @return the total file transfer time
 */
public synchronized boolean getFileTransfered(){
  notify();
  return fileTransfered;
} //end getFileTransfered() method

/**
 * Returns the control panel
 *
 * @return the control panel
 */
public synchronized ControlPanel getControlPanel(){
  notify();
  return controlPanel;
} //end getControlPanel() method

/**
 * Adds a panel to a container
 *
 * @param container the container to add the panel
 */
```

```
public void addPanel( Container container ){
  Panel dummy = new Panel();
  dummy.setBackground( BACKGROUND_COLOR );
  container.add( dummy );
} //end addPanel() method

/**
 * Saves a partial file request to file request database for future use
 *
 * @param partialFileRequest the partial file request to save in  database
 * @param partialFileBytes the partial file bytes to save in database
 */
public void savePartialFileRequest( PFTPFileRequest partialFileRequest,
                                    byte[] partialFileBytes ){

  oldPartialTransfersFiles.addElement( partialFileRequest.getFile() );
  oldPartialTransferRequests.addElement( partialFileRequest );
  oldPartialTransfersBytes.addElement( partialFileBytes );

  //oldPartialTransfersFileSizes.add(new Integer(partialFileOverallSize));

} //end savePartialFileRequest() method

/**
 * Deletes a partial file request from file request database
 *
 * @param partialFileNameRequested the name of the file of the
 *                                 request to delete from database
 */
public void deletePartialFileRequest( String partialFileNameRequested ){

  oldPartialTransferRequests.removeElement( partialFileNameRequested );
  oldPartialTransfersBytes.removeElement( partialFileNameRequested );
  oldPartialTransfersFiles.removeElement( partialFileNameRequested );

} //end deletePartialFileRequest() method

/**
 * Resets the client application
 */
public void resetApplication(){

  fileSize = 0;
  fileHash = "";
  connectionStatus = DISCONNECTED;
  connectionLossTimes = 0;
```

```
  totalFileTransferTime = 0;

  controlPanel.reset();

} //end resetApplication() method

/**
 * Starts chack for start downloading
 */
public void startCheckToAsk(){
  while( true ){
    delay( 50 );
    if( askTheFile ){
      try{
        System.out.println( "File asked" );
        askForFile( fileRequest );
      } catch( Exception e ){

      }
      askTheFile = false;
    }
  }
} //end

/**
 * Delay the execution of the code
 *
 * @param msec the delay time in milliseconds
 */
public void delay( int msec ){
  try{
    Thread.sleep( msec );
  } catch( InterruptedException ie ){
    System.out.println( "Thead sleep interupted" );
  }

} //end delay() method

/**
 * The main method
 *
 * @param args the arguments
 * @throws Exception if exception occurs in FTPClient application
 */
public static void main( String args[] ) throws Exception{
```

```
    PFTPClientPPC client = new PFTPClientPPC();
    client.startCheckToAsk();

  } //end main method

//---------------------------------------
//
//    Private Inner Classes:
//
//---------------------------------------

  /**
   * Provide a control panel to control the application
   */
  private class ControlPanel extends Panel{

    /** The PFTPClient class object */
    private PFTPClientPPC controlPanelClient;

    /** The panel that displays the image during the downloading */
    private ImageProgressPanel imagePanel;

    /** The control panel title  */
    private Panel controlPanelTitle;

    /** The file choice dialog  */
    private FileSelectionDialog fileChoiceDialog;

    /** The file choice label  */
    private Label fileChoiceLabel;

    /** The server mode choice list  */
    private java.awt.List serverModeChoiceList;

    /** The packet size choice list  */
    private java.awt.List packetSizeChoiceList;

    /** The file button group  */
    private CheckboxGroup fileButtonGroup;

    /** The server button group  */
    private CheckboxGroup serverButtonGroup;

    /**
     * The previous failed files Checkbox
     */
```

```
private Checkbox previousFailedFiles;

/**
 * The files from server Checkbox
 */
private Checkbox filesFromServer;

/**
 * The manual server selection Checkbox
 */
private Checkbox manualServerSelection;

/**
 * The predefined server selection Checkbox
 */
private Checkbox predefinedServerSelection;

/** The select server button */
private Button selectServer;

/** The button to retreive the availiable file at PFTP Server  */
private Button retreiveAvailableFiles;

/** The button to ask for the file transfer  */
private Button askTheFile;

/** The button to cancel the file transfer  */
private Button cancelTransfer;

/** The text field to enter manual the server to ask the file from  */
private TextField manualSelectServer;

/** The server IP array with servers can be chosen  */
private String[] serverIPs = {"Select server", "Auto select",
   "127.0.0.1", "131.120.105.237", "131.120.105.250", "131.120.201.46"};

/** The packet size array with the packet sizes can be chosen  */
private String[] packetSizes = {"Packet size..",
   new String( "32" ), new String( "64" ), new String( "128" ),
   new String( "256" ), new String( "512" ), new String( "1024" )};

/** The file selected  */
private String fileSelection;

/** The server selected  */
private String serverSelection;
```

```
/** The packet size selected  */
private String packetSizeSelection;

/** The boolean value if a file selected or not  */
private boolean fileSelected = false;

/** The boolean value if a server selected or not  */
private boolean serverSelected = false;

/** The boolean value if a packet size selected or not  */
private boolean packetSizeSelected = false;

/**
 * The default constructor
 *
 * @param panelClient the FTPClientVer7 object assosiated with
 * @param imagePanel the ImageProgressPanel object assosiated with
 */
public ControlPanel( PFTPClientPPC panelClient,
                     ImageProgressPanel imagePanel ){

  super();

  serverIPs = new String[VALID_SERVERS.length + 2];
  for( int i = 0; i < serverIPs.length; i++ ){
    if( i == 0 ){
      serverIPs[i] = "Select server";
    } else if( i == 1 ){
      serverIPs[i] = "Auto select";
    } else{
      serverIPs[i] = VALID_SERVERS[i - 2];
    } //end if else if
  } //end for loop

  packetSizes = new String[PACKET_SIZES.length + 1];
  for( int i = 0; i < packetSizes.length; i++ ){
    if( i == 0 ){
      packetSizes[i] = "Select packet size";
    } else{
      packetSizes[i] = PACKET_SIZES[i - 1];
    } //end if else if
  } //end for loop

  controlPanelClient = panelClient;
  imagePanel = imagePanel;
```

```
setBackground( BACKGROUND_COLOR );
setLayout( new GridLayout( 12, 2 ) );

Label controlPanelTitle = new Label( "CONTROL PANEL" );
controlPanelTitle.setFont( new Font( "Serif", Font.BOLD, 13 ) );
controlPanelTitle.setForeground( FONT_COLOR );
add( controlPanelTitle );

// just to fill the empty cells
controlPanelClient.addPanel( this );

// construct textfield with default sizing
Label fileToReceive = new Label( "File to retreive:" );
fileToReceive.setFont( new Font( "Serif", Font.BOLD, 13 ) );
fileToReceive.setForeground( FONT_COLOR );
add( fileToReceive );

fileButtonGroup = new CheckboxGroup();
previousFailedFiles = new Checkbox( "Failed transfers", fileButtonGroup, false );
previousFailedFiles.setFont( new Font( "Serif", Font.BOLD, 12 ) );
previousFailedFiles.setForeground( FONT_COLOR );
filesFromServer = new Checkbox( "Files on server", fileButtonGroup, true );
filesFromServer.setFont( new Font( "Serif", Font.BOLD, 12 ) );
filesFromServer.setForeground( FONT_COLOR );

//inner class to handle JRadioButton events
ItemListener fileMode = new ItemListener(){

  // handle Checkbox event
  public void itemStateChanged( ItemEvent event ){
    // determine whether check box selected
    if( event.getSource() == previousFailedFiles ){
      String[] fileList = new String[oldPartialTransfersFiles.size() + 1];
      for( int i = 0; i < fileList.length; i++ ){
        if( i == 0 ){
          fileList[i] = "Select file";
        } else{
          fileList[i] = ( String ) oldPartialTransfersFiles.elementAt( i -
            1 );
        } //end if else
      } //end for loop
      if( !oldPartialTransfersFiles.isEmpty() ){
        retrieveFailedTransfers = true;
        fileChoiceDialog = new FileSelectionDialog( controlPanelClient,
          controlPanelClient.getControlPanel(),
```

```
            fileChoiceLabel,
            fileList );
          packetSizeSelected = true;
        } else if( oldPartialTransfersFiles.isEmpty() ){
          MessageDialog md = new MessageDialog( controlPanelClient,
            controlPanel,
            "** Message **",
            MESSAGE_DIALOG_TYPE,
            "No file transfer failure by now!" );
          manualServerSelection.setEnabled( false );
          predefinedServerSelection.setEnabled( false );
          manualSelectServer.setEnabled( false );
          selectServer.setEnabled( false );
          serverModeChoiceList.setEnabled( false );
          retreiveAvailableFiles.setEnabled( false );
          packetSizeChoiceList.setEnabled( false );
          askTheFile.setEnabled( false );

        }
      } else if( event.getSource() == filesFromServer ){
        manualServerSelection.setEnabled( true );
        predefinedServerSelection.setEnabled( true );
        serverModeChoiceList.setEnabled( true );
        askTheFile.setEnabled( true );
      } //end if else
    } //end itemstateChanged

}; // end anonymous inner class

previousFailedFiles.addItemListener( fileMode );
previousFailedFiles.setBackground( BACKGROUND_COLOR );

filesFromServer.addItemListener( fileMode );
filesFromServer.setBackground( BACKGROUND_COLOR );

add( previousFailedFiles );

controlPanelClient.addPanel( this );

add( filesFromServer );

Label serverToReceiveFrom = new Label( "Server choice:" );
serverToReceiveFrom.setFont( new Font( "Serif", Font.BOLD, 13 ) );
serverToReceiveFrom.setForeground( FONT_COLOR );
add( serverToReceiveFrom );
```

```
serverButtonGroup = new CheckboxGroup();
manualServerSelection = new Checkbox( "Manual Selection",
                              serverButtonGroup, false );
predefinedServerSelection = new Checkbox( "Existing Servers",
                                 serverButtonGroup, true );

//inner class to handle JRadioButton events
ItemListener serverListener = new ItemListener(){

  // handle Checkbox event
  public void itemStateChanged( ItemEvent event ){
    // determine whether check box selected
    if( event.getSource() == manualServerSelection ){
      manualSelectServer.setEnabled( true );
      serverModeChoiceList.setEnabled( false );
      selectServer.setEnabled( true );
    } else if( event.getSource() == predefinedServerSelection ){
      manualSelectServer.setEnabled( false );
      selectServer.setEnabled( false );
      serverModeChoiceList.setEnabled( true );
    } //end if else
  } //end itemstateChanged

}; // end anonymous inner class

manualServerSelection.addItemListener( serverListener );
manualServerSelection.setBackground( BACKGROUND_COLOR );
manualServerSelection.setFont( new Font( "Serif", Font.BOLD, 12 ) );
manualServerSelection.setForeground( FONT_COLOR );

predefinedServerSelection.addItemListener( serverListener );
predefinedServerSelection.setBackground( BACKGROUND_COLOR );
predefinedServerSelection.setFont( new Font( "Serif", Font.BOLD, 12 ) );
predefinedServerSelection.setForeground( FONT_COLOR );

add( manualServerSelection );

controlPanelClient.addPanel( this );

add( predefinedServerSelection );

Label addServerIP = new Label( "Select server IP:" );
addServerIP.setFont( new Font( "Serif", Font.BOLD, 13 ) );
addServerIP.setForeground( FONT_COLOR );
add( addServerIP );
```

```
    manualSelectServer = new TextField( "Enter server IP" );
    addServerIP.setFont( new Font( "Serif", Font.BOLD, 13 ) );
    manualSelectServer.setEditable( true );
    manualSelectServer.setEnabled( false );
    add( manualSelectServer );

    controlPanelClient.addPanel( this );

    // constructs the add server button
    selectServer = new Button( "Select the server" );
    selectServer.setFont( new Font( "Serif", Font.BOLD, PANEL_LABEL_FONT_SIZE
) );
    selectServer.setForeground( FONT_COLOR );
    selectServer.setBackground( BACKGROUND_COLOR );
    selectServer.setEnabled( false );
    selectServer.addActionListener( new ActionListener(){
      public void actionPerformed( ActionEvent event ){

        try{
          String serverIP = manualSelectServer.getText();
          InetAddress.getByName( serverIP );
          if( !retrieveFailedTransfers ){
            retreiveAvailableFiles.setEnabled( true );
          }
          packetSizeChoiceList.setEnabled( false );
          controlPanelClient.setServerMode( serverIP );
          serverSelected = true;
        } catch( UnknownHostException uhe ){
          MessageDialog md = new MessageDialog( controlPanelClient,
                                    controlPanel,
                                    "** INFORMATION **",
                                    MESSAGE_DIALOG_TYPE,
              "Select a valid server IP address!" );
        }

      } //end actionPerformed() method
    } //end ActionListener class
    ); //end addActionListener() method

    add( selectServer );

    Label chooseServerMode = new Label( "Server mode:" );
    chooseServerMode.setFont( new Font( "Serif", Font.BOLD, 13 ) );
    chooseServerMode.setForeground( FONT_COLOR );
    add( chooseServerMode );
```

```
serverModeChoiceList = new java.awt.List( 4, false );
for( int i = 0; i < serverIPs.length; i++ ){
  serverModeChoiceList.add( serverIPs[i] );
}
serverModeChoiceList.setFont( new Font( "Serif", Font.BOLD,
                              PANEL_LABEL_FONT_SIZE ) );
serverModeChoiceList.setBackground( BACKGROUND_COLOR );
serverModeChoiceList.setForeground( FONT_COLOR );
serverModeChoiceList.setEnabled( true );
serverModeChoiceList.addItemListener(

    // anonymous inner class to handle JComboBox events
    new ItemListener(){

  // handle JComboBox event
  public void itemStateChanged( ItemEvent event ){
    // determine whether check box selected
    if( event.getStateChange() == ItemEvent.SELECTED ){
      serverSelection = serverModeChoiceList.getSelectedItem();
      serverSelected = true;
      if( !retrieveFailedTransfers ){
        retreiveAvailableFiles.setEnabled( true );
      }
    }
    controlPanelClient.setServerMode( serverSelection );
  }
} // end anonymous inner class

); // end call to addItemListener
add( serverModeChoiceList );

controlPanelClient.addPanel( this );

// constructs the Retreive files button
retreiveAvailableFiles = new Button( "Retreive files.. " );
retreiveAvailableFiles.setFont( new Font( "Serif", Font.BOLD,
                                PANEL_LABEL_FONT_SIZE ) );
retreiveAvailableFiles.setForeground( FONT_COLOR );
retreiveAvailableFiles.setBackground( BACKGROUND_COLOR );
retreiveAvailableFiles.addActionListener( new ActionListener(){
  public void actionPerformed( ActionEvent event ){
    try{
      serverAvailableFiles = controlPanelClient.getServerAvailableFiles();
      retreiveAvailableFiles.setEnabled( false );
      if( !serverAvailableFiles.isEmpty() ){
        String[] fileList = new String[serverAvailableFiles.size() + 1];
```

```
        for( int i = 0; i < fileList.length; i++ ){
          if( i == 0 ){
            fileList[i] = "Select file";
          } else{
            fileList[i] = ( String ) serverAvailableFiles.elementAt( i -
               1 );
          } //end if else
        } //end for loop
        fileChoiceDialog = new FileSelectionDialog( controlPanelClient,
           controlPanelClient.getControlPanel(),
           fileChoiceLabel,
           fileList );
        packetSizeChoiceList.setEnabled( true );
      } else{
        MessageDialog md = new MessageDialog( controlPanelClient,
           controlPanel,
           "** INFORMATION **",
           MESSAGE_DIALOG_TYPE,
           "No files availiable at server" );
      } //end if else
    } catch( IOException ioe ){
      MessageDialog md = new MessageDialog( controlPanelClient,
                                  controlPanel,
                                  "** ERROR **",
                                  MESSAGE_DIALOG_TYPE,
                                  "Fail to retreive file list" );

      retreiveAvailableFiles.setEnabled( false );

    } catch( ClassNotFoundException cnfe ){
      MessageDialog md = new MessageDialog( controlPanelClient,
                                  controlPanel,
                                  "** ERROR **",
                                  MESSAGE_DIALOG_TYPE,
                                  "Unknown object received" );

    } //end try catch block

  } //end actionPerformed() method
} //end ActionListener class
); //end addActionListener() method

retreiveAvailableFiles.setEnabled( false );
add( retreiveAvailableFiles );

Label fileWhichSelected = new Label( "File selected :" );
```

```
fileWhichSelected.setFont( new Font( "Serif", Font.BOLD, 13 ) );
fileWhichSelected.setForeground( FONT_COLOR );
add( fileWhichSelected );

fileChoiceLabel = new Label( "No file selected" );
fileChoiceLabel.setFont( new Font( "Serif", Font.BOLD,
                          PANEL_LABEL_FONT_SIZE ) );
fileChoiceLabel.setBackground( BACKGROUND_COLOR );
fileChoiceLabel.setForeground( FONT_COLOR );

add( fileChoiceLabel );

controlPanelClient.addPanel( this );

packetSizeChoiceList = new java.awt.List( 4, false );
for( int i = 0; i < packetSizes.length; i++ ){
  packetSizeChoiceList.add( packetSizes[i] );
}
packetSizeChoiceList.setFont( new Font( "Serif", Font.BOLD,
                               PANEL_LABEL_FONT_SIZE ) );
packetSizeChoiceList.setBackground( BACKGROUND_COLOR );
packetSizeChoiceList.setForeground( FONT_COLOR );
packetSizeChoiceList.setEnabled( true );
packetSizeChoiceList.addItemListener(

    // anonymous inner class to handle JComboBox events
    new ItemListener(){

  // handle JComboBox event
  public void itemStateChanged( ItemEvent event ){
    // determine whether check box selected
    if( event.getStateChange() == ItemEvent.SELECTED ){
      packetSizeSelection = packetSizeChoiceList.getSelectedItem();
      System.out.println( "packet size.. " + packetSizeSelection );
      controlPanelClient.setPacketSize( new Integer( packetSizeSelection ).
                               intValue() );
      packetSizeSelected = true;
    }

  }

} // end anonymous inner class

); // end call to addItemListener

packetSizeChoiceList.setEnabled( false );
```

```
    add( packetSizeChoiceList );

    // constructs the download button
    askTheFile = new Button( "Download" );
    askTheFile.setFont( new Font( "Serif", Font.BOLD, PANEL_LABEL_FONT_SIZE )
);
    askTheFile.setForeground( FONT_COLOR );

    askTheFile.setBackground( BACKGROUND_COLOR );
    askTheFile.addActionListener( new ActionListener(){
      public void actionPerformed( ActionEvent event ){
        if( fileSelected && serverSelected && packetSizeSelected ){
          filesFromServer.setEnabled( false );
          previousFailedFiles.setEnabled( false );
          manualServerSelection.setEnabled( false );
          predefinedServerSelection.setEnabled( false );
          manualSelectServer.setEnabled( false );
          selectServer.setEnabled( false );
          serverModeChoiceList.setEnabled( false );
          retreiveAvailableFiles.setEnabled( false );
          packetSizeChoiceList.setEnabled( false );
          askTheFile.setEnabled( false );
          cancelTransfer.setEnabled( true );

          controlPanelClient.setOffset( 0 );
          fileSelection = fileChoiceLabel.getText();

          if( !retrieveFailedTransfers ){
            fileRequest = new PFTPFileRequest( FILE_REQUEST_TYPE,
                                    fileSelection,
                                    new String( "" ), 0,
                                    new Integer(
              packetSizeSelection ).
                                    intValue() );
          } else{
            fileRequest = ( PFTPFileRequest )
              oldPartialTransferRequests.elementAt(
              oldPartialTransfersFiles.indexOf( fileSelection ) );
          }

          try{
            controlPanelClient.setAskTheFile( true );
            System.out.println( "Download pressed" );

          } catch( Exception e ){
            MessageDialog md = new MessageDialog( controlPanelClient,
```

```
        controlPanel,
        "** ERROR **",
        MESSAGE_DIALOG_TYPE,
        "Problem in asking the file" );

   } //end try catch block

 } else if( !fileSelected ){
   if( serverSelected && packetSizeSelected ){
     MessageDialog md = new MessageDialog( controlPanelClient,
        controlPanel,
        "** INFORMATION **",
        MESSAGE_DIALOG_TYPE,
        "Select server!" );

   } else if( !serverSelected && !packetSizeSelected ){
     MessageDialog md = new MessageDialog( controlPanelClient,
        controlPanel,
        "** INFORMATION **",
        MESSAGE_DIALOG_TYPE,
        "Select server,file,packet size!" );

   } else if( !serverSelected ){
     MessageDialog md = new MessageDialog( controlPanelClient,
        controlPanel,
        "** INFORMATION **",
        MESSAGE_DIALOG_TYPE,
        "Select file and server!" );

   } else if( !packetSizeSelected ){
     MessageDialog md = new MessageDialog( controlPanelClient,
        controlPanel,
        "** INFORMATION **",
        MESSAGE_DIALOG_TYPE,
        "Select file and packet size!" );

   } //end if else if
 } else if( serverSelected && !packetSizeSelected ){
   MessageDialog md = new MessageDialog( controlPanelClient,
                              controlPanel,
                              "** INFORMATION **",
                              MESSAGE_DIALOG_TYPE,
                              "Select packet size!" );

 } else if( !serverSelected && packetSizeSelected ){
   MessageDialog md = new MessageDialog( controlPanelClient,
```

```
                                    controlPanel,
                                    "** INFORMATION **",
                                    MESSAGE_DIALOG_TYPE,
                                    "Select server !" );

    } else if( !serverSelected && !packetSizeSelected ){
      MessageDialog md = new MessageDialog( controlPanelClient,
                                    controlPanel,
                                    "** INFORMATION **",
                                    MESSAGE_DIALOG_TYPE,
          "Select server and packet size!" );

    } //end if else if
  } //end actionPerformed() method
} //end ActionListener class
); //end addActionListener() method

add( askTheFile );

// constructs the cancel transfer button
cancelTransfer = new Button( "Cancel Download" );
cancelTransfer.setFont( new Font( "Serif", Font.BOLD,
                          PANEL_LABEL_FONT_SIZE ) );
cancelTransfer.setForeground( FONT_COLOR );
cancelTransfer.setBackground( BACKGROUND_COLOR );
cancelTransfer.setEnabled( false );
cancelTransfer.addActionListener( new ActionListener(){
  public void actionPerformed( ActionEvent event ){
    MessageDialog md = new MessageDialog( controlPanelClient,
                                    controlPanel,
                                    "** WARNING **",
                                    YES_NO_DIALOG_TYPE,
                                    "Are you sure ?" );
    if( messageOption == YES_OPTION ){
      controlPanelClient.cancelFileTransfer();
      imageProgressPanel.setVisible( false );
      resetApplication();
      messageOption = NO_OPTION;
    } //end if

  } //end actionPerformed() method
} //end ActionListener class
); //end addActionListener() method

add( cancelTransfer );
```

```
    setVisible( true );

  } //end constructor

  /**
   * Sets the fileSelected value
   *
   * @param state the the fileSelected boolean value
   */
  public void setFileSelected( boolean state ){

    fileSelected = state;

  } //end setFileSelected() method

  /**
   * Resets the panel
   */
  public void reset(){

    filesFromServer.setEnabled( true );
    filesFromServer.setState( true );
    previousFailedFiles.setEnabled( true );
    manualServerSelection.setEnabled( true );
    predefinedServerSelection.setEnabled( true );
    predefinedServerSelection.setState( true );
    manualSelectServer.setEnabled( false );
    selectServer.setEnabled( false );
    serverModeChoiceList.setEnabled( true );
    retreiveAvailableFiles.setEnabled( false );
    packetSizeChoiceList.setEnabled( false );
    askTheFile.setEnabled( true );
    cancelTransfer.setEnabled( false );
    fileSelected = false;
    serverSelected = false;
    packetSizeSelected = false;
    fileIsTransfering = false;

  } //end reset() method
} //end  ControlPanel class

/**
 * Provide a image progress panel to view the partial file transferred
 */
private class ImageProgressPanel extends Dialog{
```

```
/** The The PFTPClient object assosiated with this panel */
private PFTPClientPPC imageProgressPanelClient;

/** The image canvas  */
private PFTPCanvas imageCanvas;

/** The image canvas dimension  */
private Dimension imageCanvasDimension;

/** The updated times  */
private int updatedTimes;

/** The file name to preview  */
private String fileNameToPreview;

/**
 * Constructor
 *
 * @param client the PFTPClient object assosiated with this panel
 */
public ImageProgressPanel( PFTPClientPPC client ){
  super( client, "0% of " + client.getFileNameToAsk() +
        " received!", false );
  addWindowListener( new WindowListener(){
    public void windowOpened( WindowEvent event ){
    }

    public void windowClosed( WindowEvent event ){
      setVisible( false );
    }

    public void windowClosing( WindowEvent event ){
      setVisible( false );
    }

    public void windowIconified( WindowEvent event ){
    }

    public void windowDeiconified( WindowEvent event ){
    }

    public void windowActivated( WindowEvent event ){
    }

    public void windowDeactivated( WindowEvent event ){
    }
```

```
    } );

    imageProgressPanelClient = client;

    imageCanvas = new PFTPCanvas();
    imageCanvasDimension = null;
    imageCanvas.setBackground( BACKGROUND_COLOR );

    updatedTimes = 0;

    fileNameToPreview = null;

    setBackground( BACKGROUND_COLOR );
    setSize( 240, 270 );
    setLayout( new GridLayout( 1, 1 ) );
    add( imageCanvas );
    show();
  } //end constructor

  /**
   * Updates the iamage progress panel
   *
   * @param fileBytes the file byte array to display
   */
  public void updateImageProgress( byte[] fileBytes ){

    imageCanvasDimension = imageCanvas.getSize();
    Image image =
       Toolkit.getDefaultToolkit().createImage( fileBytes ).
       getScaledInstance(
       -1, imageCanvasDimension.height,
       Image.SCALE_FAST );

    imageCanvas.setImage( image );
    imageCanvas.update( imageCanvas.getGraphics() );

  } ///end updateImageProgress() method

} //end ImageProgress class

/**
 * The front dialog
 */
private class FrontDialog extends Dialog{
```

```
/** The dialog client */
private PFTPClientPPC dialogClient;

/** The dialog control panel */
private ControlPanel dialogControlPanel;

/** The dialog text field */
private Label dialogTextField;

/** The image canvas  */
private PFTPCanvas imageCanvas;

/** The image canvas dimension  */
private Dimension imageCanvasDimension;

/** The button to use the program  */
private Button useItButton;

/**
 * Constructor
 *
 * @param client the PFTPClientPPC object assosiated with
 */
public FrontDialog( PFTPClientPPC client ){
  super( client, "PFTP PocketPC Client 1.0", true );

  addWindowListener( new WindowListener(){
    public void windowOpened( WindowEvent event ){
    }

    public void windowClosed( WindowEvent event ){
      setVisible( false );
      System.exit( 0 );
    }

    public void windowClosing( WindowEvent event ){
      setVisible( false );
      System.exit( 0 );
    }

    public void windowIconified( WindowEvent event ){
    }

    public void windowDeiconified( WindowEvent event ){
    }
```

```
  public void windowActivated( WindowEvent event ){
  }

  public void windowDeactivated( WindowEvent event ){
  }

} );

dialogClient = client;
dialogTextField = new Label( "" );
setBackground( BACKGROUND_COLOR );
setSize( 240, 300 );

setLayout( new BorderLayout() );
dialogTextField.setBackground( BACKGROUND_COLOR );
add( dialogTextField, BorderLayout.NORTH );

imageCanvas = new PFTPCanvas();

imageCanvas.setBackground( BACKGROUND_COLOR );

byte[] imageBytes = new byte[1];
try{
  File file = new File( "pftpLogoClient.gif" );
  FileInputStream fis = new FileInputStream( file );
  imageBytes = new byte[( int ) file.length()];
  fis.read( imageBytes );
} catch( IOException ioe ){
  System.out.println( "No logo to show!" );
}
Image image;
image = Toolkit.getDefaultToolkit().createImage( imageBytes );
imageCanvas.setImage( image );

add( imageCanvas, BorderLayout.CENTER );

// constructs the use it button
useItButton = new Button( "Use it!" );
useItButton.setFont( new Font( "Serif", Font.BOLD, 20 ) );
useItButton.setForeground( FONT_COLOR );

useItButton.setBackground( BACKGROUND_COLOR );
useItButton.addActionListener( new ActionListener(){
  public void actionPerformed( ActionEvent event ){
    setVisible( false );
  } //end actionPerformed() method
```

```
    } //end ActionListener class
    ); //end addActionListener() method

    add( useItButton, BorderLayout.SOUTH );

    show();
   } //end constructor

} //end FrontDialog class

/**
 * A file selction dialog
 */
private class FileSelectionDialog extends Dialog{

  /** The dialog client */
  private PFTPClientPPC dialogClient;

  /** The dialog control panel */
  private ControlPanel dialogControlPanel;

  /** The dialog text field */
  private Label dialogTextField;

  /** The item list  */
  private String[] itemList;

  /** The dialog combo box */
  private java.awt.List dialogComboBox;

  /** dialog button  */
  private Button dialogButton;

  /** The dialog selection  */
  private String dialogSelection;

  /**
   * Constructor
   *
   * @param client the PFTPClientPPC object assosiated with
   * @param panel the ControlPanel object assosiated with
   * @param fileField the text field
   * @param list the file list
   */
  public FileSelectionDialog( PFTPClientPPC client,
                              ControlPanel panel,
```

```
                    Label fileField,
                    String[] list
                    ){
super( client, "Choose a file to transfer", true );

addWindowListener( new WindowListener(){
  public void windowOpened( WindowEvent event ){
  }

  public void windowClosed( WindowEvent event ){
    setVisible( false );
  }

  public void windowClosing( WindowEvent event ){
    setVisible( false );
  }

  public void windowIconified( WindowEvent event ){
  }

  public void windowDeiconified( WindowEvent event ){
  }

  public void windowActivated( WindowEvent event ){
  }

  public void windowDeactivated( WindowEvent event ){
  }

} );

dialogClient = client;
dialogControlPanel = panel;
dialogTextField = fileField;
itemList = list;
dialogSelection = null;
setBackground( BACKGROUND_COLOR );
setSize( 240, 200 );
this.setLocation( 0, 20 );
setLayout( new GridLayout( 5, 1 ) );

// just to fill the empty cell
add( new Panel() );

dialogComboBox = new java.awt.List( 4, false );
for( int k = 0; k < itemList.length; k++ ){
```

```
      dialogComboBox.add( itemList[k] );
    }
    dialogComboBox.setFont( new Font( "Serif", Font.BOLD,
                            PANEL_LABEL_FONT_SIZE ) );
    dialogComboBox.setBackground( BACKGROUND_COLOR );
    dialogComboBox.setForeground( FONT_COLOR );
    dialogComboBox.setEnabled( true );
    dialogComboBox.addItemListener(

      // anonymous inner class to handle JComboBox events
      new ItemListener(){

     // handle JComboBox event
     public void itemStateChanged( ItemEvent event ){
       // determine whether check box selected
       if( event.getStateChange() == ItemEvent.SELECTED ){
         dialogSelection = itemList[dialogComboBox.
             getSelectedIndex()];
       }

     }

    } // end anonymous inner class

    ); // end call to addItemListener

    add( dialogComboBox );

    add( new Panel() );

    Panel buttonPanel = new Panel();
    buttonPanel.setLayout( new GridLayout( 1, 3 ) );

    // constructs the ok button
    dialogButton = new Button( "OK" );
    dialogButton.setFont( new Font( "Serif", Font.BOLD,
PANEL_LABEL_FONT_SIZE ) );
    dialogButton.setForeground( FONT_COLOR );

    dialogButton.setBackground( BACKGROUND_COLOR );
    dialogButton.addActionListener( new ActionListener(){
      public void actionPerformed( ActionEvent event ){
        dialogTextField.setText( dialogSelection );
        dialogControlPanel.setFileSelected( true );
        setVisible( false );
      } //end actionPerformed() method
```

```
    } //end ActionListener class
    ); //end addActionListener() method

    buttonPanel.add( new Panel() );

    buttonPanel.add( dialogButton );

    buttonPanel.add( new Panel() );

    add( buttonPanel );

    add( new Panel() );

    show();
  } //end constructor

} //end FIleSelectionDialog class

/**
 * A Message dialog
 */
public class MessageDialog extends Dialog{

  /** The dialog client */
  private PFTPClientPPC dialogClient;

  /** The dialog title  */
  private String dialogTitle;

  /** The dialog text field */
  private Label dialogTextField;

  /** The dialog button  */
  private Button dialogButton;

  /** The message type  */
  private int messageType;

  /** The dialog message  */
  private String dialogMessage;

  /**
   * Constructor
   *
   * @param client the PFTPClientPPC object assosiated with
   * @param panel the ControlPanel object assosiated with
```

```
 * @param title the dialog title
 * @param type the dialog type
 * @param message the dialog message
 */
public MessageDialog( PFTPClientPPC client,
                      ControlPanel panel,
                      String title,
                      int type,
                      String message
                      ){
  super( client, title, true );

  dialogClient = client;
  messageType = type;
  dialogMessage = message;
  dialogTextField = new Label( dialogMessage );
  dialogTextField.setFont( new Font( "Serif", Font.BOLD, 13 ) );
  dialogTextField.setForeground( FONT_COLOR );
  setBackground( BACKGROUND_COLOR );
  setSize( 240, 120 );
  setLocation( 0, 80 );
  setLayout( new GridLayout( 3, 1 ) );

  addWindowListener( new WindowListener(){
    public void windowOpened( WindowEvent event ){
    }

    public void windowClosed( WindowEvent event ){
      setVisible( false );
    }

    public void windowClosing( WindowEvent event ){
      setVisible( false );
    }

    public void windowIconified( WindowEvent event ){
    }

    public void windowDeiconified( WindowEvent event ){
    }

    public void windowActivated( WindowEvent event ){
    }

    public void windowDeactivated( WindowEvent event ){
    }
```

```
    } );

    add( dialogTextField );

    add( new Panel() );

    Panel buttonPanel = new Panel();
    buttonPanel.setBackground( BACKGROUND_COLOR );
    if( messageType == MESSAGE_DIALOG_TYPE ){
     buttonPanel.setLayout( new GridLayout( 1, 1 ) );

     buttonPanel.add( new Panel() );

     // constructs the ok button
     dialogButton = new Button( "OK" );
     dialogButton.setFont( new Font( "Serif", Font.BOLD,
                              PANEL_LABEL_FONT_SIZE ) );
     dialogButton.setForeground( FONT_COLOR );

     dialogButton.setBackground( BACKGROUND_COLOR );
     dialogButton.addActionListener( new ActionListener(){
      public void actionPerformed( ActionEvent event ){
       setVisible( false );
      } //end actionPerformed() method
     } //end ActionListener class
     ); //end addActionListener() method

     buttonPanel.add( dialogButton );

     buttonPanel.add( new Panel() );

    } else if( messageType == YES_NO_DIALOG_TYPE ){
     buttonPanel.setLayout( new GridLayout( 1, 4 ) );

     buttonPanel.add( new Panel() );

     // constructs the yes button
     Button yesButton = new Button( "YES" );
     yesButton.setFont( new Font( "Serif", Font.BOLD, PANEL_LABEL_FONT_SIZE )
);
     yesButton.setForeground( FONT_COLOR );

     yesButton.setBackground( BACKGROUND_COLOR );
     yesButton.addActionListener( new ActionListener(){
      public void actionPerformed( ActionEvent event ){
```

```
            dialogClient.cancelFileTransfer();
            imageProgressPanel.setVisible( false );
            resetApplication();
            setVisible( false );
          } //end actionPerformed() method
        } //end ActionListener class
        ); //end addActionListener() method
        buttonPanel.add( yesButton );

        // constructs the no button
        Button noButton = new Button( "NO" );
        noButton.setFont( new Font( "Serif", Font.BOLD, PANEL_LABEL_FONT_SIZE )
);
        noButton.setForeground( FONT_COLOR );

        noButton.setBackground( BACKGROUND_COLOR );
        noButton.addActionListener( new ActionListener(){
          public void actionPerformed( ActionEvent event ){
            setVisible( false );
          } //end actionPerformed() method
        } //end ActionListener class
        ); //end addActionListener() method
        buttonPanel.add( noButton );

        buttonPanel.add( new Panel() );

      } else{

      } //end if else if

      add( buttonPanel );

      show();

    } //end constructor
  } //end MessageDialog class
} //end PFTPClientPPC class
```

```
//packages for network componets
import java.io.*;
import java.net.*;

//packages for user interface components
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import javax.swing.*;
import javax.swing.text.*;

//package for security components
import java.security.*;

//package for charecter encoding components
import sun.misc.*;

/**
 * File PFTPServer.java
 *
 * This is the PFTP Application Server
 *
 * This server is waiting for clients to connect on port and it supports
 * only file tranfer commands sit gets a file transfer command, it checks if the file
 * version exists looking at its hash valuesends a message
 * about how many bytes is the file and then send them one byte at the
 * time. To make the file look big, I placed a pause() during the each
 * byte transfer.
 * To manage the lost connection and keep track of the file tranfer
 * progress each of them, I used a data stucture to store them
 *
 * @author LT Periklis Pantoleon
 * @version 1.0
 */
public class PFTPServer extends JFrame{

//----------------------------------------
//
//    Private Data Members:
//
//----------------------------------------

  /** The default port the FTP server is waiting for clients */
  private static final int DEFAULT_PORT = 6789;
```

```
/** The timeout period for the sockets */
private static final int SO_TIME_OUT = 3000;

/** The length in bytes of the file hash value */
private static final int MD5_HASH_LENGTH = 16;

/** The key for the hashing computation*/
private static final byte[] HASH_KEY = ( new String( "Periklis" ) ).getBytes();

/** The font size for the panels' titles */
private static final int PANEL_TITLE_FONT_SIZE = 17;

/** The font size for the panels' labels */
private static final int PANEL_LABEL_FONT_SIZE = 13;

/** The option to add a file to file database */
private static final int ADD_FILE_OPTION = 1;

/** The option to remove a file to file database */
private static final int REMOVE_FILE_OPTION = 2;

/** The request type when a file is asked */
private static final int FILE_REQUEST_TYPE = 1;

/** The request type when a file array is asked */
private static final int FILE_ARRAY_REQUEST_TYPE = 0;

/** The panels' font color */
private static final Color FONT_COLOR = Color.white;

/** The color of the backgroung of the frames and panels */
private static final Color BACKGROUND_COLOR = new Color( 77, 92 , 240 );

/** The color of the panels' border */
private static final Color PANEL_BORDER_COLOR = Color.blue;

/** The file name of the logo image of the server frame */
private static final String LOGO_IMAGE = "pftpLogoServer.gif";

/** The server socket */
private ServerSocket serverSocket;

/** The socket that is assigned to the client */
private Socket mainSocket;

/** The file input stream */
```

```
  private FileInputStream fis;

  /** The map that store the file available for transfer */
  private AvailableFileMap fileMap;

  /** The list of file names in the file map */
  private ArrayList fileNamesInMap;

  /** The boolean value that indicates the listening status of the server */
  private boolean startListening;

  /** The boolean value that indicates if user want to exit the application */
  private boolean exitApplication;

  /** The number of current server clients */
  private int numberOfCurrentClients;

  /** The number of total file transfers tries */
  private int numberOfTotalTransfers;

  /** The number of failed file transfers */
  private int numberOfTransferFailures;

  /** The percentage of failed file transfers */
  private double percentageOfTransferFailures;

  /** The number of files in the server file database */
  private int numberOfLocalFiles;

//-------------------------------------------------
//
//    Private Data Members for the User Interface:
//
//-------------------------------------------------

  /** The panel that shows the server status */
  private PFTPServerStatusPanel statusPanel;

  /** The front label shown before starting the server */
  private JLabel startFrontLabel;

  /** The menu item to start the server */
  private JMenuItem startServerItem;

  /** The menu item to stop the server */
  private JMenuItem stopServerItem;
```

```
  /** The menu item to add a file to server's database */
  private JMenuItem addFileItem;

  /** The menu item to remove a file from server's database */
  private JMenuItem removeFileItem;

  /** The frame used to delete a file from server's database  */
  private JFrame deleteFileFrame;

//----------------------------------------
//
//    Constructor:
//
//----------------------------------------

  /**
   * Default constructor
   */
  public PFTPServer(){

    super( "PFTPServer" );

    statusPanel = new PFTPServerStatusPanel( this );

    fileMap = new AvailableFileMap();

    // set up Server menu and its menu items
    JMenu fileMenu = new JMenu( "Server" );
    fileMenu.setMnemonic( 'S' );

    // set up start server item
    startServerItem = new JMenuItem( "Start server" );

    startServerItem.addActionListener(

        // anonymous inner class to handle menu item event
        new ActionListener(){

      public void actionPerformed( ActionEvent event ){

        if( fileMap.isEmpty() ){
          JOptionPane.showMessageDialog( null,
             "You must add files to server database \n" +
                                " before you start the server!!",
                                "Can't start",
```

```
                        JOptionPane.INFORMATION_MESSAGE );
    } else{
      try{
        // Creates a server socket with the defaut port
        serverSocket = new ServerSocket( DEFAULT_PORT );
        startServerItem.setEnabled( false );
        stopServerItem.setEnabled( true );
        startListening = true;
        startFrontLabel.setVisible( false );
        statusPanel.setVisible( true );
        getContentPane().add( statusPanel, BorderLayout.CENTER );
      } catch( IOException ioe ){
        JOptionPane.showMessageDialog( null, ioe,
                                "Problem starting the server",
                                JOptionPane.ERROR_MESSAGE );
      } //end try catch block
    } //end if else

  } //end actionPerformed() method

} // end anonymous inner class

); // end call to addActionListener

fileMenu.add( startServerItem );

// set up stop server item
stopServerItem = new JMenuItem( "Stop server" );
stopServerItem.setEnabled( false );
stopServerItem.addActionListener(

    // anonymous inner class to handle menu item event
    new ActionListener(){

  public void actionPerformed( ActionEvent event ){

    stopServerItem.setEnabled( false );
    startServerItem.setEnabled(true);
    startListening = false;
    startFrontLabel.setVisible( true );
    statusPanel.setVisible( false );
    try{
      System.out.println("Server stopped!");
      serverSocket.close();
    }catch (IOException ioe){
      JOptionPane.showMessageDialog( null, ioe,
```

```
                                "Server can not stop, you can exit ",
                                JOptionPane.ERROR_MESSAGE );
    }//end try catch block


  } //end actionPerformed() method

} // end anonymous inner class

); // end call to addActionListener

fileMenu.add( stopServerItem );

// set up Exit menu item
JMenuItem exitItem = new JMenuItem( "Exit" );

exitItem.addActionListener(

    // anonymous inner class to handle exitItem event
    new ActionListener(){

  // terminate application if user select the exit item
  public void actionPerformed( ActionEvent event ){

    if( JOptionPane.showConfirmDialog( null,
        "Are you sure you want to exit the application?",
                                null,
                                JOptionPane.YES_NO_OPTION ) ==
       JOptionPane.OK_OPTION ){
      JOptionPane.showMessageDialog( null, "Thank you for using PFTPServer",
                               "PFTPServer",
                               JOptionPane.INFORMATION_MESSAGE );

      exitApplication = true;
      System.exit( 0 );
    } //end if

  } //end actionPerformed() method

} // end anonymous inner class

); // end addActionListener

fileMenu.add( exitItem );

// create menu bar and attach it to the server's frame
```

```
JMenuBar bar = new JMenuBar();
setJMenuBar( bar );

bar.add( fileMenu );

// set up Database menu and its menu items
JMenu databaseMenu = new JMenu( "Database" );
fileMenu.setMnemonic( 'D' );

// set up add file menu item
addFileItem = new JMenuItem( "Add file to database" );

addFileItem.addActionListener(

   // anonymous inner class to handle menu item event
   new ActionListener(){

  //check if the file extension is a valid one and add
  //the file to server's database
  public void actionPerformed( ActionEvent event ){

   File file = selectFile( ADD_FILE_OPTION );
   if( file.equals( null ) ){
     JOptionPane.showMessageDialog( null, "No file added",
                                    "PFTPServer",
                                    JOptionPane.INFORMATION_MESSAGE );
   } else if( !file.getName().endsWith( ".gif" ) &&
              !file.getName().endsWith( ".GIF" ) &&
              !file.getName().endsWith( ".png" ) &&
              !file.getName().endsWith( ".PNG" ) &&
              !file.getName().endsWith( ".jpeg" ) &&
              !file.getName().endsWith( ".jpeg" ) ){
     JOptionPane.showMessageDialog( null, "No file added \n" +
        "Please select only .gif, .png or .jpeg files",
        "PFTPServer",
        JOptionPane.INFORMATION_MESSAGE );

   } else {

     boolean fileExists = fileMap.hasFile(file.getName());
     int userOption = JOptionPane.YES_OPTION;
     if (fileExists){
     userOption = JOptionPane.showOptionDialog(null, " The file already exists!.\n"+
        "Do you want to overwrite it?", "File exists!", JOptionPane.YES_NO_OPTION,
        JOptionPane.INFORMATION_MESSAGE, null, null, null);
     }else{
```

```
      //numberOfLocalFiles++;
      }//end if

      if (userOption == JOptionPane.YES_OPTION){
        try{

          FileInputStream fis = new FileInputStream( file );
          byte[] fileBytes = new byte[( int ) file.length()];
          //read the bytes from the file
          fis.read( fileBytes );
          if (fileExists){
            fileMap.removeFile(file.getName());
            fileNamesInMap.remove(file.getName());
          }
          fileMap.addFile( file.getName(), fileBytes );
          fileNamesInMap.add( file.getName() );
          fis.close();
          numberOfLocalFiles++;

        } catch( IOException ioe ){
          JOptionPane.showMessageDialog( null, "No file added \n" +
                                  "Problem reading the file!",
                                  "Error",
                                  JOptionPane.ERROR_MESSAGE );
        } //end try catch block
      }//end if
    } //end if else if block
    statusPanel.updateServerStatus();
  } //end actionPerformed() method
} // end anonymous inner class

); // end call to addActionListener

databaseMenu.add( addFileItem );

// set up remove file menu item
removeFileItem = new JMenuItem( "Remove a file from database" );

removeFileItem.addActionListener(

    // anonymous inner class to handle menu item event
    new ActionListener(){

  // if the file database is not empty it removes the file selected
  public void actionPerformed( ActionEvent event ){
```

```
    if( fileMap.isEmpty() ){
      JOptionPane.showMessageDialog( null, "No file available in database",
                                     "PFTPServer",
                                     JOptionPane.INFORMATION_MESSAGE );

    } else{

      String[] fileList = new String[fileNamesInMap.size() + 1];
      for( int i = 0; i < fileList.length; i++ ){
        if( i == 0 ){
          fileList[i] = "Select file to delete";
        } else{
          fileList[i] = ( String ) fileNamesInMap.get( i - 1 );
        } //end if else
      } //end for loop

      deleteFileFrame = new DeleteFileFrame( fileList );

    } //end if else
    statusPanel.updateServerStatus();
  } //end actionPerformed() method

} // end anonymous inner class
); // end call to addActionListener

// create a menu item and attach it to menu bar
databaseMenu.add( removeFileItem );
bar.add( databaseMenu );

//create the front label of the main frame
startFrontLabel = new JLabel( new ImageIcon( LOGO_IMAGE ) );
getContentPane().setBackground( BACKGROUND_COLOR );
getContentPane().add( startFrontLabel, BorderLayout.CENTER );

setSize( 350, 200 );

setLocation( 300, 250 );
setVisible( true );
setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );

//intialize the main variables
startListening = false;
exitApplication = false;
fileMap = new AvailableFileMap();
fileNamesInMap = new ArrayList();
numberOfCurrentClients = 0;
```

```
  numberOfTotalTransfers = 0;
  numberOfTransferFailures = 0;
  percentageOfTransferFailures = 0;
  numberOfLocalFiles = 0;

  //update the server status
  statusPanel.updateServerStatus();

} // end default constructor

/**
 * Sets the number of current clients
 *
 * @param currentClients number of current clients
 */
public void setNumberOfCurrentClients( int currentClients ){
  numberOfCurrentClients = currentClients;
} //end setNumberOfCurrentClients() method

/**
 * Sets the number of total file transfers
 *
 * @param totalTransfers the number of total file transfers
 */
public void setNumberOfTotalTransfers( int totalTransfers ){
  numberOfTotalTransfers = totalTransfers;
} //end setNumberOfTotalTransfers() method

/**
 * Sets the number of failed file transfers
 *
 * @param totalFailures the number of failed file transfers
 */
public void setNumberOfTransferFailures( int totalFailures ){
  numberOfTransferFailures = totalFailures;
} //end setNumberOfTransferFailures() method

/**
 * Sets the percentage of failed transfers
 *
 * @param failurePercentage the percentage of failed transfers
 */
public void setPercentageOfTransferFailures( double failurePercentage ){
  percentageOfTransferFailures = failurePercentage;
} //end setPacketSize() method
```

```
/**
 * Sets the number of local files in server's database
 *
 * @param localFiles the number of local files in server's database
 */
public void setNumberOfLocalFiles( int localFiles ){
  numberOfLocalFiles = localFiles;
} //end setNumberOfLocalFiles() method

/**
 * Returns the number of current clients
 *
 * @return the number of current clients
 */
public synchronized int getNumberOfCurrentClients(){
  notify();
  return numberOfCurrentClients;
} //end getNumberOfCurrentClients() method

/**
 * Returns the number of total file transfers
 *
 * @return the number of total file transfers
 */
public synchronized int getNumberOfTotalTransfers(){
  notify();
  return numberOfTotalTransfers;
} //end getNumberOfTotalTransfers() method

/**
 * Returns the number of failed file transfers
 *
 * @return the number of failed file transfers
 */
public synchronized int getNumberOfTransferFailures(){
  notify();
  return numberOfTransferFailures;
} //end getNumberOfTransferFailures() method

/**
 * Returns the percentage of failed transfers
 *
 * @return the percentage of failed transfers
 */
public synchronized double getPercentageOfTransferFailures(){
  notify();
```

```
  if( getNumberOfTotalTransfers() != 0 ){
    setPercentageOfTransferFailures(
       ( double ) ( getNumberOfTransferFailures() /
                 getNumberOfTotalTransfers() ) *
       100 );
  } else{
    setPercentageOfTransferFailures( 0 );
  } //end if else
  return percentageOfTransferFailures;
} //end getPercentageOfTransferFailures() method

/**
 * Returns the number of local files in server's database
 *
 * @return the number of local files in server's database
 */
public synchronized int getNumberOfLocalFiles(){
  notify();
  return numberOfLocalFiles;
} //end getNumberOfLocalFiles() method

/**
 * Adds a panel to a container
 *
 * @param container the container to add the panel
 */
public void addPanel( Container container ){
  JPanel dummy = new JPanel();
  dummy.setBackground( BACKGROUND_COLOR );
  container.add( dummy );
} //end addPanel() method

/**
 * Removes a file from server's database
 *
 * @param fileForDeletion the string file name to delete from database
 */
public void removeFromDatabase( String fileForDeletion ){
  if( !fileForDeletion.equals( null ) ){
    fileMap.removeFile( fileForDeletion );
    fileNamesInMap.remove( fileForDeletion );
    numberOfLocalFiles--;
    JOptionPane.showMessageDialog( null,
                          "File " + fileForDeletion + " deleted!",
                          "PFTPServer",
                          JOptionPane.INFORMATION_MESSAGE );
```

```
  } else{
    JOptionPane.showMessageDialog( null, "No file deleted!",
                                   "PFTPServer",
                                   JOptionPane.INFORMATION_MESSAGE );
  } //end if else
} //end removeFromDatabase() method

/**
 * Returns the file that selected by the user to be added or removed
 *
 * @return the file that selected by the user to be added or removed
 * @param addOrRemoveOption the integer option for add or remove
 */
private File selectFile( int addOrRemoveOption ){
  File selectedFileName = null;
  try{
    // display file dialog so user can select file
    JFileChooser fileChooser = new JFileChooser();
    fileChooser.setFileSelectionMode(
       JFileChooser.FILES_ONLY );

    int result;

    if( addOrRemoveOption == ADD_FILE_OPTION ){
      result = fileChooser.showDialog( this, "Add" );
    } else{
      result = fileChooser.showDialog( this, "Remove" );
    }

    // if user clicked Cancel button on dialog, return
    if( result == JFileChooser.CANCEL_OPTION ){
      return null;
    } else if( result == JFileChooser.APPROVE_OPTION ){

      // obtain selected file
      selectedFileName = fileChooser.getSelectedFile();

    } //end if else if
  } catch( NullPointerException npe ){

  } //end try catch block

  return selectedFileName;

} // end selectFile() method
```

```
/**
 * Delays the current thread by certain amount of milliseconds
 *
 * @param  msec the milliseconds to delay the execution
 */
public void delay( int msec ){
  try{
    Thread.sleep( msec );
  } catch( InterruptedException ie ){
    System.out.println( "Thead sleep interupted" );
  } // end try catch block
} // end delay() method

/**
 * Manage the starting and stoping of the server side
 */
public void startListening(){
  while( !exitApplication ){
    delay( 50 );

    if( startListening ){
      System.out.println( "Server started!" );
      try{
        while( startListening ){
          mainSocket = serverSocket.accept();
          ClientHandler newClientHandler = new ClientHandler( mainSocket );
          System.out.println( "New connection established" );
          statusPanel.updateServerStatus();
          Thread newConnection = new Thread( newClientHandler );
          newConnection.start();
          if( !startListening ){
            startServerItem.setEnabled( true );
          }
        } //end while loop


      } catch( IOException ioe ){
        if( startListening ){
          JOptionPane.showMessageDialog( null, ioe,
                                         "Server can not start listening " +
                                         "in port " + DEFAULT_PORT,
                                         JOptionPane.ERROR_MESSAGE );
        }//end if
      } //end try catch block
```

```
    } else if (serverSocket != null && !serverSocket.isClosed()){

    }// end if else
   } //end outer whole loop
   System.exit( 0 );
  } //end startListening method

  /**
   * Main method to start
   *
   * @param args argument list
   * @throws Exception
   */
  public static void main( String args[] ){

    //Initialiaze the server application
    PFTPServer theFTPServer = new PFTPServer();
    //start listening at default port
    theFTPServer.startListening();

  } //end main method

//---------------------------------------
//
//    Private Inner Classes:
//
//---------------------------------------

  /**
   * Handles the connection and the file transfer with each client
   */
  private class ClientHandler implements Runnable{

    /** The socket to comunicate with the client */
    private Socket clientSocket;

    /** The object input stream to receive objects from client */
    private ObjectInputStream inObjectFromClient;

    /** The object output stream to send objects to client */
    private ObjectOutputStream outObjectToClient;

    /** The file request received from client */
    private PFTPFileRequest request;

    /** The vector of the files available in the server database */
```

```
private Vector filesAvailable;

/** The request type */
private int requestType;

/** The requested file name */
private String fileName;

/** The hash value of the requested file */
private String fileHash;

/** The packet size */
private int packetSize;

/** The offset value in the file request */
private int offset;

/** The length of the requested file in bytes */
private int numOfBytes;

/**
 * Constructor
 *
 * @param aSocket the socket to communicate with the client
 * @throws SocketException
 */
public ClientHandler( Socket aSocket ) throws SocketException{
  // Initialize the socket to client
  clientSocket = aSocket;
  // Sets the timout period of the socket
  clientSocket.setSoTimeout(SO_TIME_OUT);
} //end constructor

/**
 * The run() method of the Runnable ClientHandler class that
 * handles the connection to the client
 */
public void run(){

  setNumberOfCurrentClients( getNumberOfCurrentClients() + 1 );
  System.out.println("Clients: " + getNumberOfCurrentClients() );
  statusPanel.updateServerStatus();
  // Opens the streams for communicating with the client
  openStreams();
  // Handles the file transfer
  handleFileRequest();
```

```
  // Closes the connection when done
  closeConnection();
  // Reduce the number of clients connected
  setNumberOfCurrentClients( getNumberOfCurrentClients() - 1 );
  System.out.println("Clients: " + getNumberOfCurrentClients() );
  statusPanel.updateServerStatus();

} // end of run() method

/**
 * Opens the streams for communicating with the client
 */
private void openStreams(){

  try{

    outObjectToClient = new ObjectOutputStream(
       clientSocket.getOutputStream() );

    inObjectFromClient = new ObjectInputStream(
       clientSocket.getInputStream() );

  } catch( IOException ioe ){

    System.out.println( "Server: Failed to open I/O streams\n" + ioe );
    System.exit( 1 );

  } // end try catch block
} // end openStreams() method

/**
 * Handles the file request of the client
 */
private void handleFileRequest(){

  try{

    System.out.println( "New client connected!!" );
    request = ( PFTPFileRequest ) inObjectFromClient.readObject();
    requestType = request.getRequestType();

    if( requestType == FILE_ARRAY_REQUEST_TYPE ){
      filesAvailable = new Vector();
      for( int k = 0; k < fileNamesInMap.size(); k++ ){
        filesAvailable.add( fileNamesInMap.get( k ) );
      } //end for loop
```

```
 outObjectToClient.writeObject( filesAvailable );
 outObjectToClient.flush();
} else if( requestType == FILE_REQUEST_TYPE ){
 clientSocket.setSoTimeout( SO_TIME_OUT );
 fileName = request.getFile();
 fileHash = request.getHash();
 offset = request.getOffset();
 packetSize = request.getPacketSize();

 System.out.println( "Received request packet -> ... File name: " +
                fileName +
                "\n                    File hash: " +
                fileHash +
                "\n                      Offset: " +
                offset +
                "\n                  Packet size: " +
                packetSize );

 if( offset == 0 ){
  fileHash = getHashValue( fileName );
  if( fileHash.equals( new String( "fileNotFound" ) ) ){
   System.out.println( "File " + fileName + " not found!" );
   outObjectToClient.writeUTF( fileHash );
   outObjectToClient.flush();
  } else if( fileHash.equals( new String( "hashProblem" ) ) ){
   System.out.println( "Problem in hashing file " + fileName );
   outObjectToClient.writeUTF( fileHash );
   outObjectToClient.flush();
  } else{
   System.out.println( "File found!" );

   outObjectToClient.writeUTF( fileHash );
   outObjectToClient.flush();
   System.out.println( "Hash " + fileHash + " is send, length " +
                fileHash.length() );
   System.out.println( "Sending the file...." );
   sendFile( fileName, packetSize, offset );
  } //end if else if
 } else{
  try{
   if( checkHash( fileName, fileHash ) ){
    System.out.println( "Sending the file...." );
    outObjectToClient.writeUTF( "Start sending" );
    sendFile( fileName, packetSize, offset );
   } else{
    outObjectToClient.writeUTF( "fileNotMatch" );
```

```
        } //end if else
       } catch( IOException ioe ){
        System.out.println( "Problem in sending the file!" );
        setNumberOfTransferFailures( getNumberOfTransferFailures() + 1 );
        statusPanel.updateServerStatus();
        System.out.println( ioe );
       } // end try catch block
     } // end if else block
    } else{
     outObjectToClient.writeUTF( "noValidRequest" );
     outObjectToClient.flush();
    } //end if else if
  } catch( Exception e ){
    System.out.println( e );
  }
} // end HandleFileRequest() method

/**
 * Closes the socket connection
 */
private void closeConnection(){
  try{
    clientSocket.close();
  } catch( IOException ioe ){
  } //end try catch block
} //end closeConnection() method

/**
 * Compute the hash value of a given file
 *
 * @param fileName the name of the file
 * @return the String representation of the file hash value
 */
public String getHashValue( String fileName ){

  String theHash = null;
  try{
    File theFile = new File( fileName );
    fis = new FileInputStream( theFile );
    byte[] buffer = new byte[8192];
    int length;
    //Select the MD5 hash algorithm
    MessageDigest md = MessageDigest.getInstance( "MD5" );
    while( ( length = fis.read( buffer ) ) != -1 ){
      md.update( buffer, 0, length );
    }
```

```
    fis.close(); // Close the file stream to free the file

    // Final computation of the hash
    byte[] hash = md.digest();

    // Create a String version of the hash value using the default
    // 64bit Character encoding algorithm.
    BASE64Encoder encoder = new BASE64Encoder();
    theHash = encoder.encode( hash );

   } catch( FileNotFoundException fnfe ){ // In case file not found
    System.out.println( fnfe );
    theHash = new String( "fileNotFound" );
   } catch( Exception e ){ // In case there is problem
    System.out.println( e ); // with hash computation
    theHash = new String( "hashProblem" );
   }

  return theHash;

} // end getHashValue() method

/**
 * Checks if a file is asked have the given hash value
 *
 * @param file the name of the file
 * @param hashValueToCheck the hash value to check the validity
 * @return true if hashes match and false if they don't
 */
public boolean checkHash( String file, String hashValueToCheck ){
  boolean check;
  String existedHash = getHashValue( file );
  if( existedHash.equals( hashValueToCheck ) ){ //check the two hashes
    check = true;
  } else{
    check = false;
  } //end if else

  return check;

}//end checkHash() method

/**
 * Sends the file to client with the asked packet size and offset
 *
 * @param  nameOfFile the name of the file
```

```
 * @param  packetSize the hash value to check the validity
 * @param  offset the offset from where client wants the file
 * @throws IOException if something goes wrong with the stream use
 * @throws FileNotFoundException if the file asked doesn't exist
 */
public void sendFile( String nameOfFile, int packetSize, int offset ) throws
    IOException, FileNotFoundException{
  File file = new File( nameOfFile );
  int numOfBytes = ( int ) file.length();
  int numOfPackets = ( numOfBytes / packetSize ) + 1;
  System.out.println( nameOfFile );

  //sending the file total number of bytes
  outObjectToClient.writeInt( numOfBytes );
  outObjectToClient.flush();
  System.out.println( "File size sent " + numOfBytes + " bytes" );

  // open a sream to the file
  FileInputStream inFile = new FileInputStream( nameOfFile );
  byte[] fileInBytes = new byte[numOfBytes + packetSize];

  //read the bytes from the file
  inFile.read( fileInBytes );

  System.out.println( "Client: " +
                  ( clientSocket.getInetAddress() ).toString()
                  + " from port " + clientSocket.getPort()
                  + " asked for file " + nameOfFile
                  + "\nOffset: " + offset );

  int currentPacket = offset;

  try{
    //send the packets
    setNumberOfTotalTransfers( getNumberOfTotalTransfers() + 1 );
    while( currentPacket < numOfPackets ){
      outObjectToClient.write( fileInBytes, currentPacket * packetSize,
                        packetSize );
      outObjectToClient.flush();
      System.out.println( "Bytes " + currentPacket * packetSize + " to " +
                      ( currentPacket + 1 ) * packetSize + " sent" );
      delay( 500 );

      currentPacket++;

    } //end while loop
```

```
      //check if the transfer ends

      if( currentPacket == numOfPackets &&
          inObjectFromClient.readBoolean() ){
        System.out.println( "File is sent!" );
      } //end if

    } catch( IOException ioe ){
      System.out.println( ioe );
      setNumberOfTransferFailures( getNumberOfTransferFailures() + 1 );
    } //end try catch block
  } // end sendFile() method
} //end ClientHandler inner class

/**
 * A panel that contains useful informations about the server
 * operation status.
 */
private class PFTPServerStatusPanel extends JPanel{

  /** The PFTP server that has this panel */
  private PFTPServer serverStatusPanelServer;

  /** The title of the status panel  */
  private JTextPane serverStatusPanelTitle;

  /** The textpane that shows the number of current clients */
  private JTextPane numberOfCurrentClientsInfo;

  /** The textpane that shows the number of total file transfers */
  private JTextPane numberOfTotalTransfersInfo;

  /** The textpane that shows the number of failed file transfers */
  private JTextPane percentageOfTransferFailuresInfo;

     /** The textpane that shows the number of local files in server's database */
  private JTextPane numberOfLocalFilesInfo;

  /**
   * Constructor
   *
   * @param server the PFTPServer that owns the panel
   */
  public PFTPServerStatusPanel( PFTPServer server ){
    super();
```

```
    serverStatusPanelServer = server;

    setBorder( BorderFactory.createLineBorder( PANEL_BORDER_COLOR ) );

    numberOfCurrentClientsInfo = getTitledPanel( "", PANEL_LABEL_FONT_SIZE,
       FONT_COLOR,
       BACKGROUND_COLOR, false, true );
    numberOfCurrentClientsInfo.setForeground( FONT_COLOR );

    numberOfTotalTransfersInfo = getTitledPanel( "", PANEL_LABEL_FONT_SIZE,
       FONT_COLOR,
       BACKGROUND_COLOR, false, true );
    numberOfTotalTransfersInfo.setForeground( FONT_COLOR );

    percentageOfTransferFailuresInfo = getTitledPanel( "",
       PANEL_LABEL_FONT_SIZE,
       FONT_COLOR,
       BACKGROUND_COLOR, false, true );
    percentageOfTransferFailuresInfo.setForeground( FONT_COLOR );

    numberOfLocalFilesInfo = getTitledPanel( "",
                                PANEL_LABEL_FONT_SIZE,
                                FONT_COLOR,
                                BACKGROUND_COLOR, false, true );
    numberOfLocalFilesInfo.setForeground( FONT_COLOR );

    setBackground( BACKGROUND_COLOR );
    setLayout( new GridLayout( 5, 2 ) );

    add( getTitledPanel( "SERVER STATUS", PANEL_TITLE_FONT_SIZE,
FONT_COLOR,
                BACKGROUND_COLOR, true, true ) );

    add( getTitledPanel( "", PANEL_LABEL_FONT_SIZE, FONT_COLOR,
                BACKGROUND_COLOR, false, true ) );

    add( getTitledPanel( "Current clients :", PANEL_LABEL_FONT_SIZE,
                FONT_COLOR,
                BACKGROUND_COLOR, false, true ) );

    add( numberOfCurrentClientsInfo );

    add( getTitledPanel( "Total file transfers :", PANEL_LABEL_FONT_SIZE,
                FONT_COLOR,
                BACKGROUND_COLOR, false, true ) );
```

```
  add( numberOfTotalTransfersInfo );

  add( getTitledPanel( "Transfer failures :", PANEL_LABEL_FONT_SIZE,
                  FONT_COLOR,
                  BACKGROUND_COLOR, false, true ) );

  add( percentageOfTransferFailuresInfo );

  add( getTitledPanel( "Files in database :", PANEL_LABEL_FONT_SIZE,
                  FONT_COLOR,
                  BACKGROUND_COLOR, false, true ) );

  add( numberOfLocalFilesInfo );

  setVisible( false );
  setEnabled( false );

}

/**
 * Return a titled JTextPanel with specific attributes
 *
 * @param titleText the delay time in milliseconds
 * @param charSize the character size
 * @param charColor the character color
 * @param backgroundColor the backgroundColor
 * @param isUnderlined if the character is underlined or not
 * @param isBold if the character is bold or not
 * @return a titled JtextPanel with specific attributes
 */
public JTextPane getTitledPanel( String titleText, int charSize,
                        Color charColor, Color backgroundColor,
                        boolean isUnderlined, boolean isBold ){
  JTextPane textPane;
  DefaultStyledDocument doc = new DefaultStyledDocument();
  textPane = new JTextPane( doc );
  MutableAttributeSet attr = new SimpleAttributeSet();
  StyleConstants.setFontSize( attr, charSize );
  StyleConstants.setUnderline( attr, isUnderlined );
  StyleConstants.setForeground( attr, charColor );
  StyleConstants.setBold( attr, isBold );
  textPane.setEditable( false );
  textPane.setEnabled( false );
  textPane.setBackground( backgroundColor );
  textPane.setCharacterAttributes( attr, true );
```

```
  textPane.setText( titleText );
  return textPane;
} //end getTitledPanel() method

/**
 * Updates file transfer's status
 */
public synchronized void updateServerStatus(){
  notify();
  numberOfCurrentClientsInfo.setText( new Integer(
     serverStatusPanelServer.getNumberOfCurrentClients() ).toString() );
  numberOfTotalTransfersInfo.setText( new Integer(
     serverStatusPanelServer.getNumberOfTotalTransfers() ).toString() );
  percentageOfTransferFailuresInfo.setText( new Double(
     serverStatusPanelServer.getPercentageOfTransferFailures() ).toString() +
                                    "%" );
  numberOfLocalFilesInfo.setText( new Integer(
     serverStatusPanelServer.getNumberOfLocalFiles() ).toString() );
} //end updateFileTransferStatus() method

/**
 * Resets the panel
 */
public void reset(){

  SwingUtilities.invokeLater(
     new UpdateTextPane( numberOfCurrentClientsInfo, "" ) );

  SwingUtilities.invokeLater(
     new UpdateTextPane( numberOfTotalTransfersInfo, "" ) );

  SwingUtilities.invokeLater(
     new UpdateTextPane( percentageOfTransferFailuresInfo, "" ) );

  SwingUtilities.invokeLater(
     new UpdateTextPane( numberOfLocalFilesInfo, "" ) );

} //end reset() method

/**
 * Updates a TextPane
 */
public class UpdateTextPane extends Thread{

  /** The text pane to update  */
  private JTextPane textPane;
```

```
    /** The the text to set to the text pane  */
    private String text;

    /**
     * Constructor
     *
     * @param jTextPane the text pane to update
     * @param aText the text to be shown on the text pane
     */
    public UpdateTextPane( JTextPane jTextPane, String aText ){
      textPane = jTextPane;
      text = aText;
    } //end constructor

    // method called to update outputArea
    public void run(){
      textPane.setText( text );
    } //end run() method

  } // end class UpdateThread

}

/**
 * Updates a TextPane
 */
private class DeleteFileFrame extends JFrame{

  /** The combo box with files to select  */
  private JComboBox filesToDelete;

  /** The string array with file names available to delete  */
  private String[] fileNames;

  /** The button to use for deleting  */
  private JButton deleteButton;

  /** The file name selected to be deleted  */
  private String fileSelected;

  /**
   * Constructor
   *
   * @param theFileNames the frame's string array with files names to delete
   */
```

```
public DeleteFileFrame( String[] theFileNames ){

  super( "Delete file" );

  // get content pane and set its layout
  Container container = getContentPane();
  container.setLayout( new GridLayout( 4, 1 ) );

  fileNames = theFileNames;
  fileSelected = null;

  addPanel(container);

  // set up delete file JComboBox
  filesToDelete = new JComboBox( fileNames );
  filesToDelete.setBackground( BACKGROUND_COLOR );
  filesToDelete.setForeground( FONT_COLOR );
  filesToDelete.addItemListener(

     // anonymous inner class to handle JComboBox events
     new ItemListener(){

   // handle JComboBox event
   public void itemStateChanged( ItemEvent event ){
     // determine whether check box selected
     if( event.getStateChange() == ItemEvent.SELECTED )
       fileSelected = fileNames[filesToDelete.getSelectedIndex()];
     deleteButton.setEnabled( true );
   }//end itemStateChanged() method

  } // end anonymous inner class

  ); // end call to addItemListener

  container.add( filesToDelete );

  addPanel(container);

  // set up delete button
  deleteButton = new JButton( "Delete file" );
  deleteButton.setEnabled( false );
  deleteButton.setBackground( BACKGROUND_COLOR );
  deleteButton.setForeground( FONT_COLOR );
  deleteButton.addActionListener( new ActionListener(){
    public void actionPerformed( ActionEvent event ){
      removeFromDatabase( fileSelected );
```

```
          setVisible( false );
        } //end actionPerformed() method
      } //end ActionListener class
      ); //end addActionListener() method

      container.add( deleteButton );

      setSize( 250, 200 );
      setLocation( 340, 250 );
      setResizable( false );
      setDefaultCloseOperation( JFrame.HIDE_ON_CLOSE );
      setVisible( true );
    } //end constructor

  } // end class DeleteFileFrame

} // end PFTPServer class
```

```
//packages for transmiting the object over
//an input and output stream
import java.io.Serializable;

/**
 * File PFTPFileRequest.java
 *
 * Provides an object to hold the data of the file transfer request
 *
 * @author LT Periklis Pantoleon
 * @version 1.0
 */
public class PFTPFileRequest
    implements Serializable {

  /** The type of the file request */
  private int requestType;

  /** The file to request */
  private String file;

  /** The hash value of the file */
  private String hash;

  /** The byte offset */
  private int offset;

  /** The socket to comunicate with the client */
  private int packetSize;

  /**
   * Constructor
   *
   * @param theRequestType the type of the request 1 if it is file request and
   *                                    0 if it is available files request
   * @param theFileName the file name
   * @param theFileHash the file hash value
   * @param theFileOffset the file offset
   * @param thePacketSize the file transfer packet size
   */
  public PFTPFileRequest(int theRequestType,
                         String theFileName,
                         String theFileHash,
                         int theFileOffset,
                         int thePacketSize) {
```

```
  this.requestType = theRequestType;
  this.file = theFileName;
  this.hash = theFileHash;
  this.offset = theFileOffset;
  this.packetSize = thePacketSize;

} //end FileRequest class constructor

/**
 * Gets the request type
 *
 * @return the request type
 */
public int getRequestType() {
  return requestType;
} // end getRequestType() method


/**
 * Gets the file name
 *
 * @return the file name
 */
public String getFile() {
  return file;
} // end getFile() method

/**
 * Gets the file hash value
 *
 * @return the file hash value
 */
public String getHash() {
  return hash;
} // end getHash() method

/**
 * Gets the offset
 *
 * @return the offset
 */
public int getOffset() {
  return offset;
} // end getOffset() method
```

```
/**
 * Gets the packet size
 *
 * @return the packet size
 */
public int getPacketSize() {
  return packetSize;
} // end getPacketSize() method

/**
 * Sets the request type
 *
 * @param newRequestType the request type
 */
public void setRequestType(int newRequestType) {
  this.requestType = newRequestType;
} // end setRequestType() method

/**
 * Sets the file name
 *
 * @param newFile the file name
 */
public void setFile(String newFile) {
  this.file = newFile;
} // end setFile() method

/**
 * Sets the file hash value
 *
 * @param newHash the file hash value
 */
public void setHash(String newHash) {
  this.hash = newHash;
} // end setHash() method

/**
 * Sets the file offset
 *
 * @param newOffset the file offset
 */
public void setOffset(int newOffset) {
  this.offset = newOffset;
} // end setOffset() method

/**
```

```
   * Sets the file transfer packet size
   *
   * @param newPacketSize the file tranfer packet size
   */
  public void setPacketSize(int newPacketSize) {
    this.packetSize = newPacketSize;
  } // end getPacketSize() method

  /**
   * Checks for equality
   *
   * @param request the file request that we want to check the equality with
   * @return the true if the requests aren equal, false if they aren't
   */
  public boolean equals(PFTPFileRequest request) {
    boolean isEqual = false;
    if (this.getFile() == request.getFile() &&
        this.getHash() == request.getHash() &&
        this.getOffset() == request.getOffset() &&
        this.getPacketSize() == request.getPacketSize()) {
      isEqual = true;
    }
    return isEqual;
  } // end equals() method

} // end FileRequest class
```

```
//packages
import java.io.*;
import java.util.*;

/**
 * File AvailableFileMap.java
 *
 * This the class that is used to store the availiable to download
 * files in the PFTP server.
 *
 * @author LT Periklis Pantoleon
 * @version 1.0
 */

public class AvailableFileMap
    extends HashMap
    implements Serializable {

  /** The capacity of the file map */
  private int capacity;

  /**
   * Default Constructor
   */
  public AvailableFileMap() {

    super();
    capacity = 16;

  } //end AvailableFileMap class default constructor

  /**
   * Constructor
   *
   * @param theCapacity the file map capacity
   */
  public AvailableFileMap(int theCapacity) {

    super();
    capacity = theCapacity;

  } //end AvailableFileMap class constructor

  /**
   * Returns the file map capacity
```

```
 *
 * @return the file map capacity
 */
public int getCapacity() {

  return capacity;

} // end getCapacity() method

/**
 * Sets the file map capacity
 *
 * @param newCapacity the new file map capacity
 */
public void setCapacity(int newCapacity) {

  capacity = newCapacity;

} // end setCapacity() method

/**
 * Check if there is an entry with a specific file name as key
 *
 * @param fileName the string file name key value
 * @return hasFile true if there is an entry, false otherwise
 */
public boolean hasFile(String fileName) {

  boolean hasFile = this.containsKey(fileName);
  return hasFile;

} // end hasFile() method

/**
 * Returns the file byte array that mas the key file name
 *
 * @param fileName the string file name key value
 * @return the file byte array that mas the key file name
 */
public byte[] getFileBytes(String fileName) {

  byte[] fileBytes = (byte[]) this.get(fileName);
  return fileBytes;

} // end getFileBytes() method
```

```
 /**
  * Add a file byte array and assosiates it with a file name
  *
  * @param fileName the file name of the file key value
  * @param fileBytes the file bytes of the key file
  */
 public void addFile(String fileName, byte[] fileBytes) {

   this.put(fileName, fileBytes);

 } // end addFile() method

 /**
  * Removes a file byte array that assosiated with a file name
  *
  * @param fileName the file name to remove from file map
  */
 public void removeFile(String fileName) {

   this.remove(fileName);

 } // end removeFile() method

} // end AvailableFileMap class
```

```
//packages
import java.awt.Canvas;
import java.awt.Image;
import java.awt.Graphics;
import java.awt.Dimension;

/**
 * File PFTPCanvas.java
 *
 * This the class that draws a Canvas with an image on it
 *
 * @author LT Periklis Pantoleon
 * @version 1.0
 */
public class PFTPCanvas extends Canvas{

  /** The image on the canvas */
  private Image canvasImage = null;

  /**
   * Default Constructor
   */
  public PFTPCanvas(){
    super();
  } //end PFTPCanvas class default constructor

  /**
   * Sets the image on the canvas
   *
   * @param anImage the new image on the canvas
   */
  public void setImage( Image anImage ){
    canvasImage = anImage;
  } // end setImage() method

  /**
   * Returns the image on the canvas
   *
   * @return the image on the canvas
   */
  public Image getImage(){
    return canvasImage;
  } // end setImage() method

  /**
```

```
   * Updates the image on the canvas
   *
   * @param g the Graphics object of the canvas
   */
  public void update( Graphics g ){
    paint( g );
  } // end update() method

  /**
   * Draws the image on the canvas
   *
   * @param g the Graphics object of the canvas
   */
  public void paint( Graphics g ){
    if( canvasImage != null ){
      Dimension size = getSize();
      g.clearRect( 0, 0, size.width, size.height );
      g.drawImage( canvasImage, 0, 0, this );
    } //end if
  } // end paint() method
} // end PFTPCanvas class
```

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

[1] Abhay Bhushant, *"The File Transfer Protocol"*, RFC 172, MIT, June 1971.

[2] J. Postel and J. Reynolds, *"The File Transfer Protocol"*, RFC 959, ISI, October 1985.

[3] Abhay Bhushant, *"Data Transfer Protocol"*, RFC 171, MIT, June 1971.

[4] K. R. Sollins, *"The TFTP Protocol (Revision 2)"*, RFC 783, MIT, June 1981.

[5] K. Sollins, *"The TFTP Protocol (Revision 2)"*, RFC 1350, MIT, July 1992.

[6] Mark K. Lottor, *"Simple File Transfer Protocol"*, RFC 913, September 1984.

[7] Network Working Group, *"The Hypertext Transfer Protocol – HTTP1.1"*, RFC 2616, June 1999.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Peter Denning, Chairman, Code CS
   Department of Computer Science
   Naval Postgraduate School
   Monterey, California

4. Professor Wen Su, Code CS
   Department of Computer Science
   Naval Postgraduate School
   Monterey, California

5. John Gibson
   Department of computer Science
   Naval Postgraduate School
   Monterey, California

6. Lieutenant Periklis K. Pantoleon
   Greek Navy, Greece